Al-Azhar University
Faculty of Engineering
Electronics & communication Department

B.Sc. Graduation Project
(Electronics & communications)

Supervised By

Dr. Mohamed Yasin Ibrahim Afifi

ADVANCED DRIVER ASSISTANCE SYSTEM

BASED ON V2V AND V2X

2023

Presented By

Mohamed Ramadan Mohamed
Bahgat Mahmoud Bahgat
Abdullah Mohamed Abdullah
Mariam Ahmed Saeed
Amira Mahmoud Badawy Mahmoud

Al-Azhar University
Faculty of Engineering
Electronics & communication Department

B.Sc. Graduation Project
(Electronics & communications)

Supervised By

Dr. Mohamed Yasin Ibrahim Afifi

ADVANCED DRIVER ASSISTANCE SYSTEM

BASED ON V2V AND V2X

2023

Presented By

Mohamed Ramadan Mohamed
Bahgat Mahmoud Bahgat
Abdullah Mohamed Abdullah
Mariam Ahmed Saeed
Amira Mahmoud Badawy Mahmoud

Acknowledgement

Our primary thanks go to Allah for the Success in the Completion this work.

Foremost, I would like to express my sincere gratitude to our supervisor, ***Dr. Mohamed Yasin***, for the continuous support of our graduation project and research, and for his patience, motivation, enthusiasm, and immense knowledge. We could not have imagined having a better advisor and mentor for our graduation project.

Besides our supervisor, we would like to thank ***Eng. Ahmed Assaf*** , ***Eng. Motasem El-Sayed*** , ***Eng. Mohammed Zaghlol*** , ***Eng. Mazen Osama*** for being so patient and asking our difficult questions and guiding us to achieve great results.

Our appreciation to all members of the electrical engineering department at Al-Azhar University.

Last but not the least, we would like to thank our family: our parents supported us spiritually throughout my life.

Abstract

Traffic accidents are highly undesirable for road users, yet they are quite common. Most unfortunately we didn't learn from our mistakes along the way. Most road users are aware of the general rules and safety measures when using the road, but only the negligence of some road users causes accidents and accidents. The main cause of accidents and incidents is human error. We explain some common human behaviors that lead to accidents.

99% of road accidents are due to human error. It is an inconvenient fact that most drivers continue to make the same mistakes on the road without noticing their poor driving habits.

Many accidents occur due to distracted driving and human error. Partially automated driver assistance systems (ADAS) can help drivers avoid collisions and maintain control of the vehicle by issuing warning signals.

Partially automated driver assistance systems (ADAS) can help drivers avoid collisions and maintain control of the vehicle by issuing warning signals.

We will use V2V communication between cars
V2V communication enables vehicles to wirelessly exchange information about their speed, location and route.

We will use V2X communication between my car and traffic lights
V2X, this new field includes vehicle-to-vehicle (V2V) communication as well as vehicle-to-thing communication.

We will explain the wireless communication between the drive subsystem and the vehicle, the hardware components in each subsystem (drive subsystem, processing subsystem, and acceleration subsystem), the function of each subsystem, and the data exchanged between them.

We will explain the wireless communication between two vehicles on the RPI using WIFI, the hardware components on the RPI (camera, servos), the functions and tasks of the RPI, and the serial communication between subsystems.

We will explain the functions and tasks of the RPI in relation to the ambulance, the functions and tasks of the STM in relation to street intersections, and the serial communication between the RPI and the STM.

Abbreviations

ADAS	Advanced Driver Assistance System
API	Application Programming interface
BMP	Bitmap
CRC	Cyclic Redundancy check
CT	Computer tomography
GCC	GNU compiler collection
GPIO	General purpose input/output
GPU	Graphics processing unit
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HDMI	High definition multimedia interface
HQ	High quality
IR	InfraRed
IOS	IPhone operating system
IOT	Internet of things
IP	Internet protocol
I2C	Inter-Integrated Circuit
Linux	Lovable Intellect Not Using XP
MCAL	Microcontroller Abstraction Layer
MP	megapixel
MRI	Magnetic resonance imaging
PWM	Pulse Width Modulation
RTC	Real Time Clock
OpenCV	Open source computer vision
OS	Operating system
RAM	Random access memory
RISC	Reduced instruction set computer
RPI	Raspberry pi

SCP	Secure copy protocol
SD	Secure digital
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SVM	Support vector machine
TCP	Transmission control protocol
UART	Universal Asynchronous Receive Transmit
UDP	User datagram protocol
UNIX	UNiplexed Information Computing System
USB	Universal serial bus
V2P	Vehicle to Person
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
V2X	Vehicle to Anything
Wi-Fi	Wireless fidelity
YOLO	You only look once

Contents

Acknowledgement 2
Abstract 3
Abbreviations 4
Contents 6
List of figures 10
List of tables 13
Objectives 14
Expected Outcome 15
Research Methodology, Implementation and Validation 15
Project Architecture 16
Module A: Automotive system 17
Module B: Linux system 17
Module C: Traffic lights system 17
Automotive system 18
Chapter 1: Processing subsystem 19
1.1 what is the processing system 19
1.2 System input data 21
1.2.1 Data from raspberry pi MCU 21
1.2.2 Data from Bluetooth module 22
1.2.3 Data from ultrasonic sensors 22
1.3 Subsystem Cases 24
1.3.1 blind spots 25
1.3.2 forward moving 27
1.3.3 Backward moving 31
1.3.4 Right & Left Case 35
1.4 Raspberry pi Role in Vehicle-to Vehicle 36
1.4.1 Introduction 36
1.4.2 Vehicle-to-Vehicle (V2V) 36
1.4.3 Raspberry Pi 37
1.4.4 Task 1 38
1.4.4.a Description 38
1.4.4.b Hardware Components 39
1.4.5 Task 2 40
1.4.5.a Description 40
1.4.5.b Blind Spot When Driving a Car On The Road 41
Chapter 2: Acceleration subsystem 43
2.1 Driving Mobile Vehicle 43
2.1.1 Hardware Component and Controllers Used 44
2.1.1.a Controllers 45
2.1.1.b Actuators 48

2.1.1.c Power System 52
2.2 Software Design 54
2.2.1 Abstraction Layers Design 54
2.2.2 Internal SW Architecture APIS 54
2.2.2.a GPIO APIs description 55
2.2.2.b RCC APIs description 56
2.2.2.c NVIC APIs description 56
2.2.2.d SPI APIs description 56
2.2.2.e UART APIs description 57
2.2.2.f Timers APIs description 57
2.2.2.g Systic APIs description 57
2.2.3 External SW Architecture APIS 58
2.2.3.a LED APIs description 58
2.2.3.b TFT APIs description 58
Linux system 60
Chapter 3: Communication subsystem 62
3.1 Wire communication 62
3.1.1 USART 62
3.1.1.a UART main features 62
3.1.1.b UART Specs 62
3.1.1.c UART Hardware interface 63
3.1.1.d USART functional description 63
3.1.1.e USART Transmitter 66
3.1.1.f USART Receiver 69
3.1.1.g USART Overrun error 71
3.1.1.h USART Fractional baud rate generation 72
3.1.1.i USART Parity control 72
3.1.1.j USART Interrupts 73
3.1.2 SPI 75
3.1.2.a SPI Hardware 76
3.1.2.b main features of the SPI protocol 76
3.1.2.c SPI Specs 77
3.1.2.d SPI Error detects 77
3.1.2.e SPI Data frame 78
3.1.2.f SPI Advantage 79
3.1.2.g SPI Usage 80
3.2 Wireless communication 81
3.2.1 Bluetooth module 81
3.2.1.a Bluetooth Usage 82
3.2.1.b How Bluetooth Works 83
3.2.1.c Bluetooth system Blocks 83
3.2.1.d Bluetooth technology 84

3.2.1.e Bluetooth Advantages 85
3.2.1.f Bluetooth Implementation 85
3.2.1.g Frequency hopping 86
3.2.2 infrared (IR) module 87
3.2.2.a IR Communication 88
3.2.2.b IR Transmitter 89
3.2.2.b IR Receiver 89
3.2.2.c Nec Protocol 90
3.2.3 WIFI module 91
3.2.3.1 Raspberry pi and WIFI 91
3.2.3.1.a Why use WIFI to communicate between raspberry pis ...91
3.2.3.1.b Setting up your raspberry pis 92
3.2.3.2 Socket programming 92
3.2.3.2.a What is socket programming 92
3.2.3.2.b Client-server communication 92
3.2.3.3 Methodology 93
3.2.3.3.a General steps 93
3.2.3.3.b Hardware list 94
3.2.3.3.c Software list 95
3.2.3.4 Client-Server Communication within Localhost 95
3.2.3.4.a Coding 95
Chapter 4: Embedded Linux subsystem 99
4.1 Introduction 99
4.2 Linux 99
4.2.1 Linux History 100
4.2.2 Why Linux 101
4.2.3 The difference between Linux an Windows 102
4.3 Embedded Linux 102
4.4 Embedded Linux Architecture 103
4.4.1 Toolchain 104
4.4.2 Bootloader 106
4.4.3 Kernel 110
4.4.4 Root Filesystem 111
4.4.5 Application 112
4.5 Build Systems 113
4.6 The Yocto Project 114
Chapter 5: OpenCV subsystem 118
5.1 Who uses openCV 118
5.2 Computer Vision 119
5.2.1 What is computer vision 119
5.2.2 What is image processing 119
5.2.3 Computer vision vs Image processing 120

5.3 Applications of computer vision 120
5.3.1 Transportation 120
5.3.1.a Self-driving cars 120
5.3.1.b Pedestrian detection 121
5.3.1.c traffic flow analysis 122
5.3.2 Traffic flow analysis medical imaging 122
5.4 OpenCV Library 123
5.4.1 Features of OpenCV library 123
5.5 What is needed for installation 124
5.6 Python interface for OpenCV 124
5.7 Handling Files, Cameras, and GUIs 125
5.7.1 Reading/writing an image file 125
5.7.2 Reading/writing on a video 126
5.7.3 Camera with openCV 128
5.8 Haar cascade and YOLO 130
Traffic lights system 132
Chapter 6: Traffic lights with Ambulance system 133
6.1 Methodology 134
6.2 Number of the ways that can be used to
determine the traffic signal for an ambulance 134
6.2.1 Using Camera 134
6.2.2 Using wireless communication technology 136
6.2.3 Using the Sensor and Wireless communication together 138
6.2.4 Using smart internet and cloud technology 140
6.3 Hardware connects 141
6.4 Examples of cities that have successfully implemented these systems ...142
Summary 144
Future Work 146
Reference 148

List of figures

Chapter 1		
Figure 1.1	Central Processing Unit	Page | 19
Figure 1.2	The Engine Control Module (ECM)	Page | 20
Figure 1.3	STM32F104"Black Pill"	Page | 21
Figure 1.4	Raspberry pi interfacing with UART	Page | 21
Figure 1.5	STM32F401 interfacing with Bluetooth	Page | 22
Figure 1.6	Ultrasonic Distance Sensor	Page | 23
Figure 1.7	Ultrasonic sensor positions	Page | 23
Figure 1.8	Blind spot case	Page | 25
Figure 1.9	Blind Spot problem	Page | 25
Figure 1.10	Detect blind spot using ultrasonic sensor	Page | 26
Figure 1.11	Displaying the blind spot to the driver	Page | 26
Figure 1.12	Blind spot Flowchart	Page | 27
Figure 1.13	Safe distance with forward direction	Page | 28
Figure 1.14	Forward direction	Page | 28
Figure 1.15	Forward direction with unsafe distance	Page | 29
Figure 1.16	Forward direction with obstacle	Page | 29
Figure 1.17	Flow chart of GO direction	Page | 30
Figure 1.18	Backward moving	Page | 31
Figure 1.19	Autonomous Parking	Page | 32
Figure 1.20	Assistance Parking	Page | 32
Figure 1.21	Flowchart of back case	Page | 34
Figure 1.22	Right case	Page | 35
Figure 1.23	Raspberry Pi 3B+	Page | 38
Figure 1.24	Raspberry pi UART	Page | 39
Chapter 2		
Figure 2.1	Acceleration Hardware	Page | 44
Figure 2.2	Arduino uno pinout	Page | 46

Figure 2.3	Pinout of STM32F401	Page | 48
Figure 2.4	DC Motors Types	Page | 49
Figure 2.5	DC Motors H-Bridge	Page | 49
Figure 2.6	H-Bridge Circuit	Page | 50
Figure 2.7	PWM Duty cycle	Page | 51
Figure 2.8	Battery Parallel Connector	Page | 52
Figure 2.9	Connections for 12V Battery Pack with BMS	Page | 53
Figure 2.10	Architecture layer of acceleration subsystem	Page | 54
Chapter 3		
Figure 3.1	UART Hardware interface	Page | 63
Figure 3.2	UART Block diagram	Page | 64
Figure 3.3	Word length programming	Page | 65
Figure 3.4	Configurable stop bits	Page | 67
Figure 3.5	TC/TXE behavior when transmitting	Page | 68
Figure 3.6	Start bit detection when oversampling by 16 or 8	Page | 69
Figure 3.7	USART interrupt mapping diagram	Page | 74
Figure 3.8	SPI pinout	Page | 75
Figure 3.9	SPI data frame	Page | 78
Figure 3.10	Block diagram of bluetooth subsystem	Page | 82
Figure 3.11	Bluetooth system Blocks	Page | 83
Figure 3.12	Bluetooth piconet scatternet scenarios	Page | 84
Figure 3.13	master to slave connection in a piconet	Page | 86
Figure 3.14	Wavelength in manometer	Page | 87
Figure 3.15	IR module	Page | 88
Figure 3.16	IR Transmitter	Page | 89
Figure 3.17	IR Receiver	Page | 90
Figure 3.18	Nec frame	Page | 90
Figure 3.19	WIFI communication between two raspberry pis	Page | 91
Figure 3.20	Client-server communications	Page | 93
Figure 3.21	Raspberry Pi 3 B+	Page | 94

Figure 3.22	SD card	Page | 94
Figure 3.23	Wireless router	Page | 94
Chapter 4		
Figure 4.1	GNU/Linux OS	Page | 101
Figure 4.2	Embedded Linux Architecture	Page | 103
Figure 4.3	Types of Toolchain	Page | 104
Figure 4.4	Cross-compilation toolchain	Page | 105
Figure 4.5	Booting Sequence	Page | 106
Figure 4.6	Raspberry pi boot sequence	Page | 109
Figure 4.7	Kernel main jobs	Page | 110
Figure 4.8	Yocto project components	Page | 115
Chapter 5		
Figure 5.1	Self-driving cars	Page | 120
Figure 5.2	Pedestrian detection	Page | 121
Figure 5.3	Traffic flow analysis	Page | 122
Chapter 6		
Figure 6.1	Detect cars using camera	Page | 135
Figure 6.2	V2X wireless communication technology	Page | 136
Figure 6.3	Traffic signal with RFID reader	Page | 138
Figure 6.4	Smart internet and cloud technology	Page | 140
Figure 6.5	Raspberry pi with motor	Page | 141
Figure 6.6	Raspberry pi with Camera	Page | 141
Figure 6.7	Arduino with IR sensor	Page | 141

List of tables

Chapter 1		
Table 1.1	Safe distance between two cars	Page | 35
Chapter 2		
Table 2.1	Technical Features arduino uno	Page | 46
Table 2.2	Connection with the BMS	Page | 53
Chapter 3		
Table 3.1	Frame format	Page | 72
Table 3.2	USART interrupt requests	Page | 73
Chapter 4		
Table 4.1	Linux vs Windows	Page | 102
Table 4.2	Architectures of different bootloaders	Page | 110

Introduction

0

The number of road traffic accidents is one of the biggest social problems in the world today. The World Health Organization estimates that 1.2 million people die and as many as 50 million are injured each year . In Germany, more than 200,000 accidents occur each year in built-up areas at speeds below 30 km/h. The German Insurance Association reports that approximately 50 percent of rear-end collisions occur without braking, and 70 percent of rear-end collisions occur due to insufficient braking strength. Many accidents could have been prevented if automated systems had assisted people in braking. Advanced Driver Assistance Systems (ADAS) cannot completely prevent accidents, but they can do a better job of protecting us from some of the human elements that account for most road accidents [1].

Our project is an advanced driver assistance system (ADAS), which is a robot car that has subsystems to deal with all driving situations including avoiding risks and warning driver.

We implement this project using STM32F401, Raspberry pi, Arduino using three modules :

- Module A: Automotive system
- Module B: Linux system
- Module C: Traffic lights system

Objectives

1. Prevent deaths and injuries by reducing the number of car accidents and the serious impact of those that cannot be avoided.
2. Reduce traffic fatalities and injuries by providing advanced warning systems, automated braking, lane departure warnings, blind-spot monitoring, and more.
3. Prevent accidents by allowing vehicles in transit to send position and speed data to one another over a mesh network.
4. Make the ambulance arrive as quickly as possible using V2X system.
5. We want our project to cover all cases and needs of the user to be more secure.

Expected Outcome

The major outcome of this project is aimed to the robot can map all road conditions, control of the car, and warning the driver in case there is danger to the driver or the car.

And monitoring the road and alerting in the event that there is an ambulance and paving the traffic light and the way for it.

Research Methodology, Implementation and Validation

The methods that will use to finish this project are enlisted here.

- Learn the basic principle of Real-Time Operating systems (RTOS), then choose the appropriate one and implement it in the project to control all system requirements.

- Learn the basic principle of interfacing and coding the drivers, then implement it in the project.

- Learn the basic principle of Automotive and How car works, then implement it in the project to control of all cases of road.

- Learn the basic principle of Linux administration and interfacing with RPI , then implement it in the project to control of camera and WIFI.

- Learn the basic principle of Battery Management System (BMS), then implement it in the project to control the power of the robot and charge time.

- Programming of microcontroller to control the I/O devices.

- Circuits developments and design for the robot control unit.

- Connect all parts of the systems and test the communication between them.

- Making hardware prototype and testing the robot function.

Project Architecture

Our Project contains 3 modules:

- Module A: ADAS system.
- Module B: V2V system.
- Module C: V2X system.

We'll talk about them individually.

Module A: Automotive system

We will explain the wireless communication between Driving subsystem and the vehicle, the hardware components at each subsystem(Driving subsystem, Processing subsystem and Acceleration subsystem), the functionality of each subsystem and data shared between them.

This module consists of tow chapters, listed as:
Chapter 1: Processing subsystem
Chapter 2: Acceleration subsystem

Module B: Linux system

We will explain the wireless communication between two vehicles using WIFI at RPI, the hardware components at RPI(camera, servo), the functionality and tasks of RPI and serial communication between subsystems.

This module consists of two chapters, listed as:
Chapter 3: Communication subsystem
Chapter 4: Embedded Linux subsystem
Chapter 5: OpenCV subsystem

Module C: Traffic lights system

We will explain the functionality and tasks of RPI related to ambulance, the functionality and tasks of STM related to intersections of road and serial communication between RPI and STM.

This module consists of two chapters, listed as:
Chapter 6: Traffic lights with Ambulance system

AUTOMOTIVE SYSTEM

Chapter 1: Processing Subsystem

Chapter 2: Acceleration Subsystem

Processing Subsystem

1

In this chapter, we'll talk about processing subsystem with it's functionality, hardware components used in this subsystem, subsystem software to control of it's functionality, also we will discuss the controllers used.

This subsystem received data of direction and speed from bluetooth module then compare this data with data received from RPI then take right action to share it with accleration sudsystem.

1.1 what is the processing system:

A processing CPU (Central Processing Unit) is the primary component of a computer that performs most of the processing tasks and calculations that enable a computer to function. It is sometimes also referred to as the processor or the "brain" of the computer. The CPU receives instructions from the computer's memory and performs arithmetic and logical operations on the data according to these instructions. The speed and efficiency of the CPU can have a significant impact on the overall performance of a computer, as it determines how quickly tasks can be completed. Modern CPUs are made up of multiple cores, which allows them to perform multiple tasks simultaneously and can significantly improve performance for tasks that can be parallelized. Additionally, CPUs often have various levels of cache memory, which allows them to store frequently accessed data and instructions, reducing the time it takes to access them from the main memory [2].

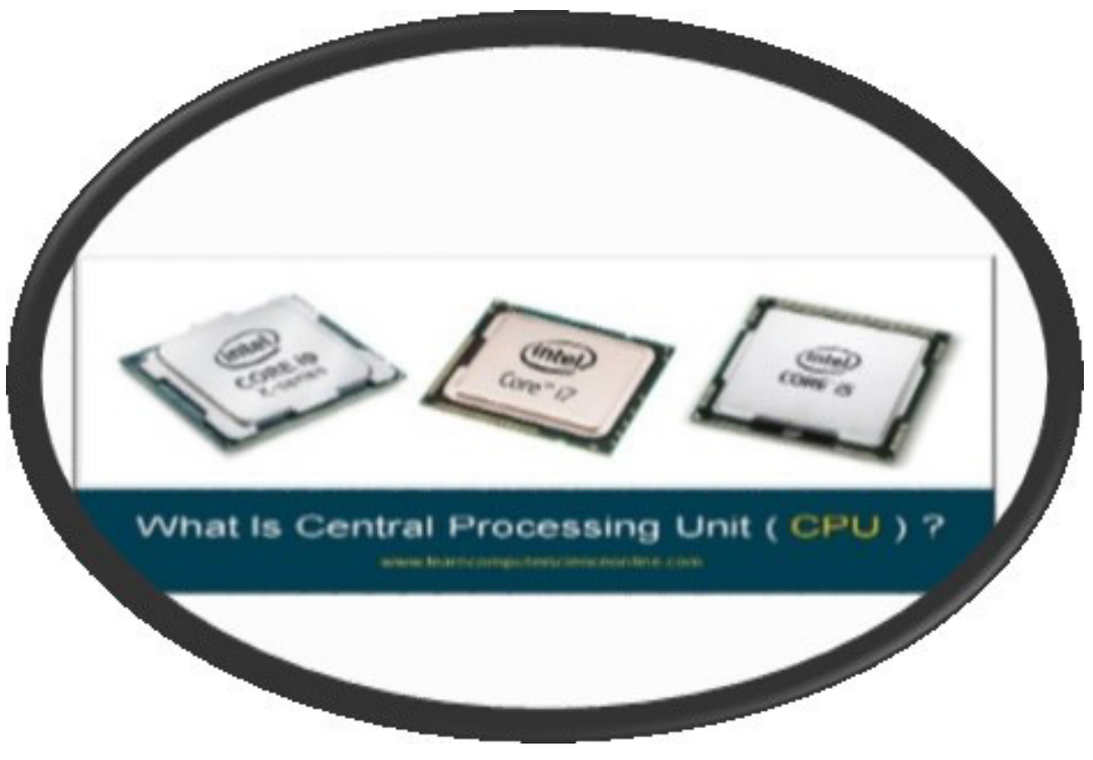


Figure 1.1: Central Processing Unit

The processing system in Automotive:- In a car, the processing CPU is often referred to as the Engine Control Module (ECM) or the Engine Control Unit (ECU). The ECM is responsible for monitoring and controlling various systems in the car, such as the engine, transmission, and emissions. The ECM receives input from various sensors placed throughout the car, including the oxygen sensor, air flow sensors, and temperature sensors. Based on this input, the ECM makes decisions about how to adjust the engine's fuel injection, ignition timing, and other parameters to ensure the car is running efficiently and with minimal emissions. Additionally, modern cars often have multiple ECUs that are responsible for controlling different systems, such as the Anti-Lock Braking System (ABS), the Electronic Stability Control (ESC), and the airbag system. These ECUs work together to ensure the car is functioning properly and safely. The processing power of the ECM and other ECUs in a car can have a significant impact on the car's performance, fuel efficiency, and overall safety. Advances in technology have allowed for more advanced and powerful ECUs, which can improve performance and efficiency while reducing emissions and improving safety features [2].

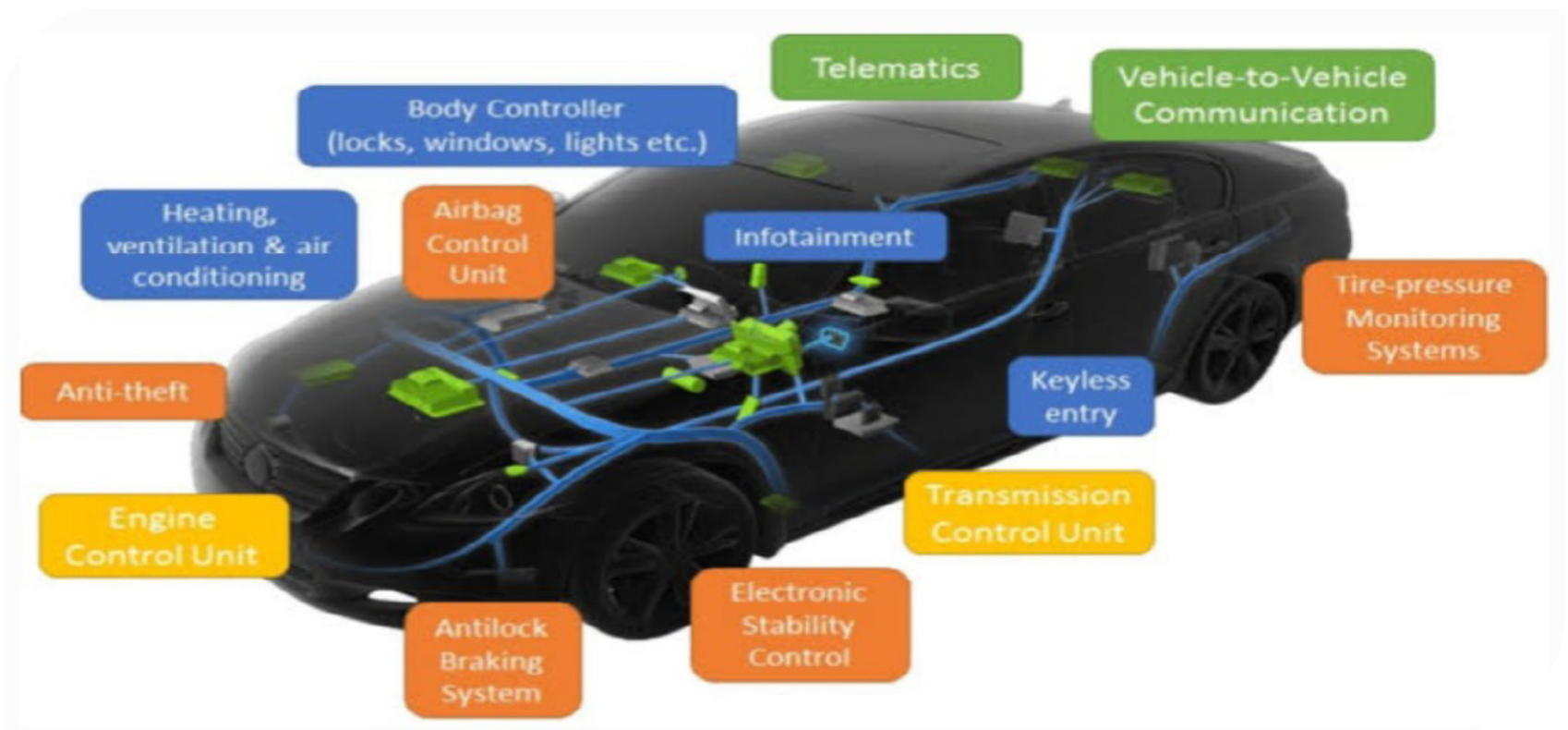


Figure 1.2: the Engine Control Module (ECM)

The processing system in this system:-

In this chapter we will explain how we process data which be sent by driving system and raspberry pi to help driver to take correct decision. There are many cases we will discuss it as blind spot and forward direction and parking. The MCU “STMF32” which be responsible to process data and it is input data is the speed of my car from Bluetooth module, the direction of my car from Bluetooth module too, the distance surrounded my car from ultrasonic sensors and the speed of car which front me from raspberry pi. Then the MCU gives the suitable action based on algorithms of code.

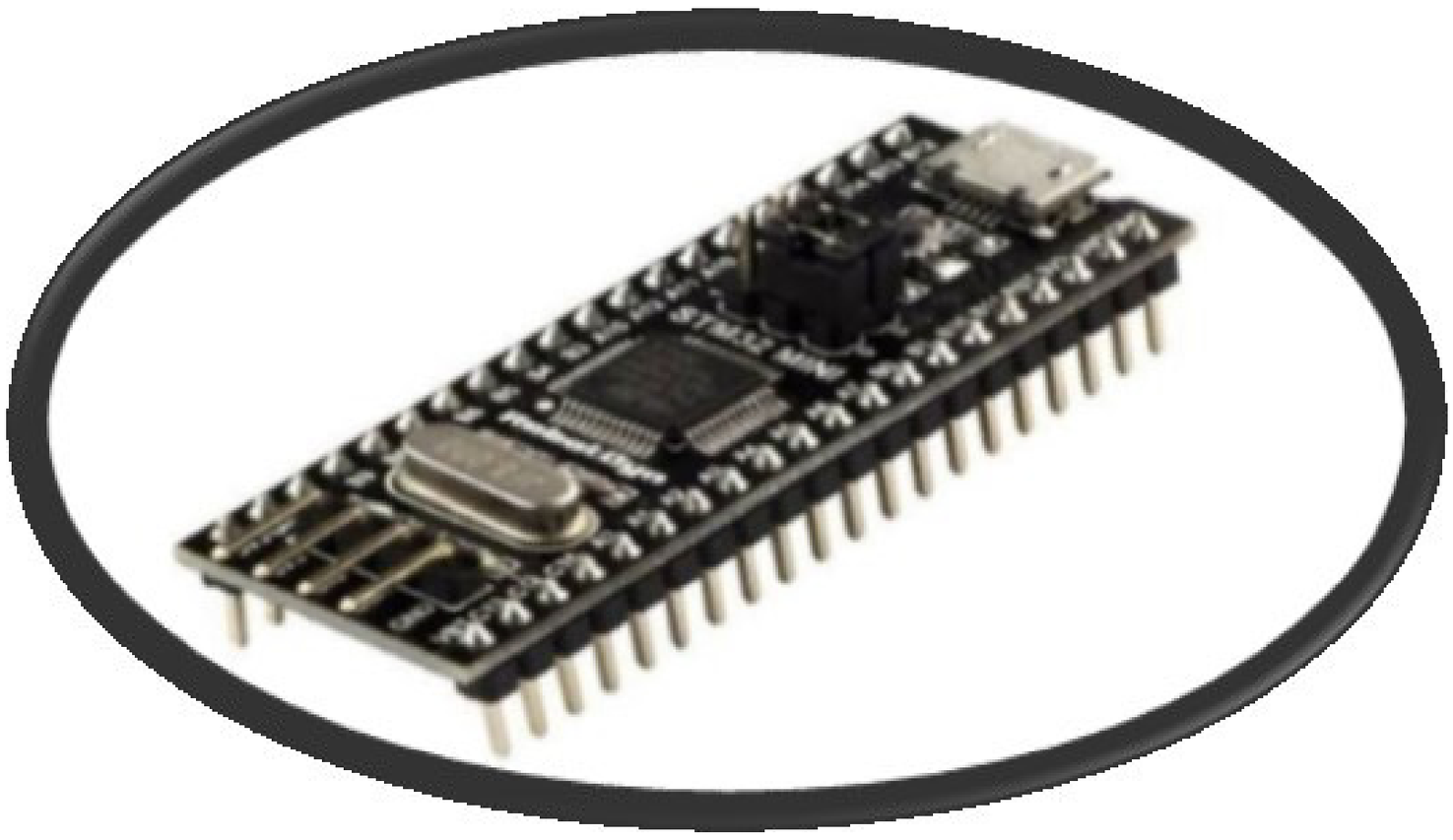


Figure 1.3: STM32F104"Black Pill"

1.2 System input data:

There are two types of input data (from RPI and from bluetooth).
Data from RPI contain Is there car at my direction or not? if yes: I received speed of this car and compare with my speed.
Data from bluetooth contain my speed and my direction

1.2.1 Data from raspberry pi MCU:

The STM receives information about neighboring cars, specifically the speed of the front car, from raspberry pi MCI through UART Communication protocol. This information is useful in the case of moving forward, where the speed of the two cars is compared and the probability of collision is determined, and accordingly the appropriate decision is taken.

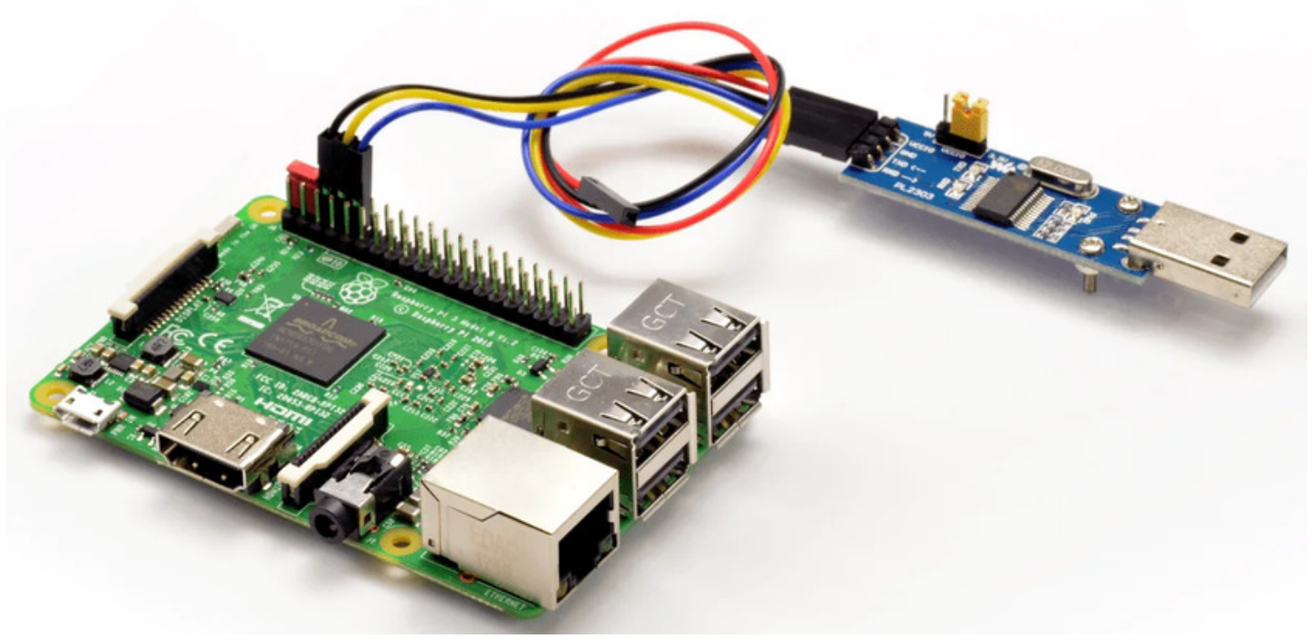


Figure 1.4: Raspberry pi interfacing with UART

1.2.2 Data from Bluetooth module:

The STM receives information about neighboring cars, specifically the speed of the front car, from raspberry pi MCI through UART Communication protocol. This information is useful in the case of moving forward, where the speed of the two cars is compared and the probability of collision is determined, and accordingly the appropriate decision is taken.

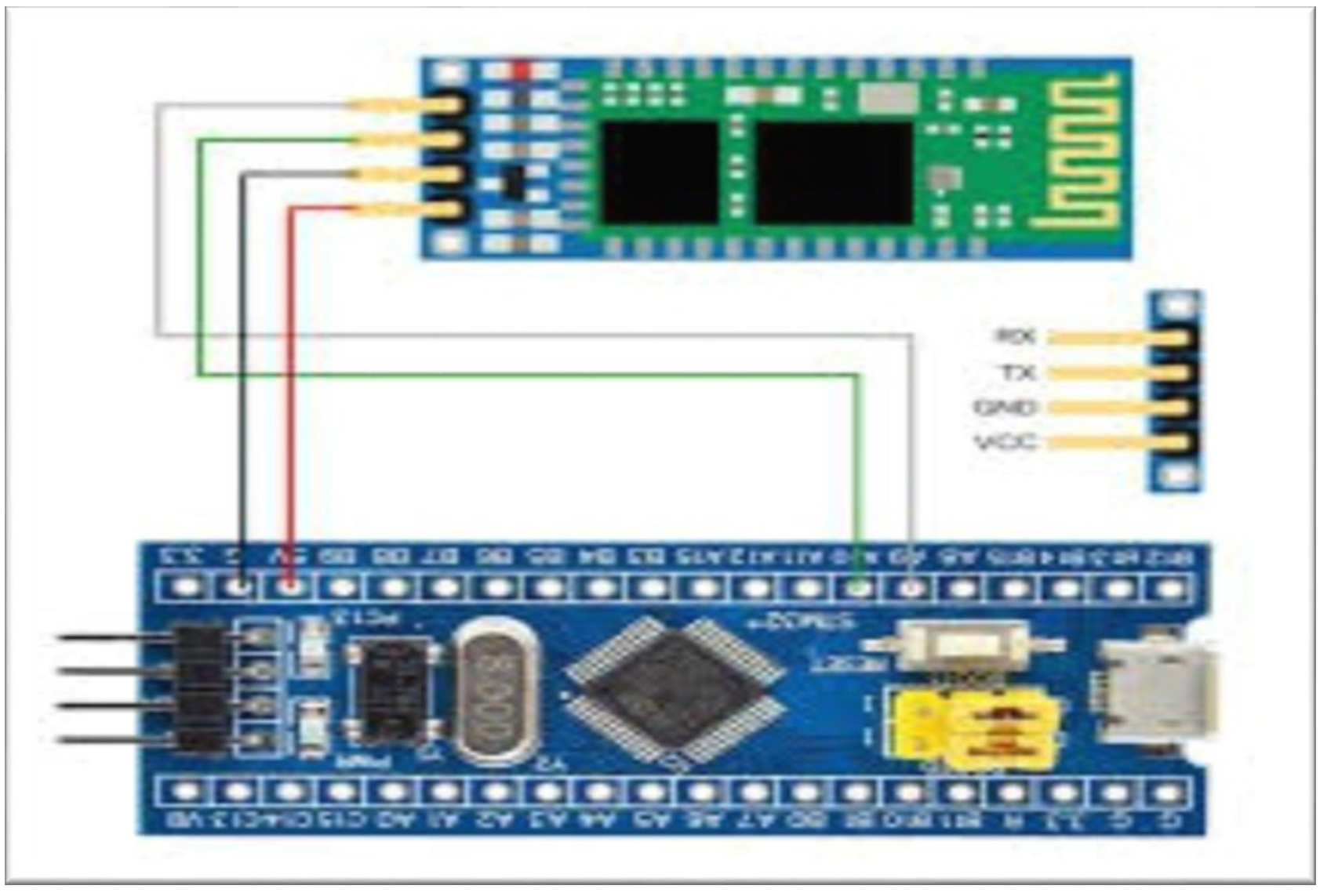


Figure 1.5: stm32f104 interfacing with Bluetooth

1.2.3 Data from ultrasonic sensors:

An ultrasonic distance sensor is a device that uses sound waves to measure the distance between the sensor and an object. It works on the principle of echolocation, which is the same principle used by bats to navigate through their surroundings.

The sensor emits high-frequency sound waves (usually above 20 kHz) from a transducer, which is a small speaker-like device. These sound waves travel through the air and bounce off of any nearby objects. The sensor then detects the reflected sound waves using a receiver, which is another transducer that is capable of detecting these high-frequency waves.

The time it takes for the sound waves to travel to the object and back to the sensor is then measured by the sensor, and this time is used to calculate the distance between the sensor and the object. This is done using the speed of sound, which is around 343 meters per second at room temperature.

In order to calculate the distance, the sensor sends out a burst of sound waves and then waits for the reflected waves to return. The time it takes for the sound waves to travel to the object and back is measured using a timer, and this time is then converted into distance using the formula: distance = (speed of sound x time) / 2 The distance is divided by 2 because the sound waves have to travel to the object and back to the sensor, so the distance is actually twice the distance that the sound waves traveled. Ultrasonic distance sensors are commonly used in robotics, automation, and other applications where it is necessary to measure distances accurately and reliably. They are relatively inexpensive and easy to use, and they can be used in a wide range of environments and conditions. However, they do have some limitations, such as the fact that they can be affected by temperature, humidity, and other environmental factors, and they may not work well in noisy or cluttered environments [3].

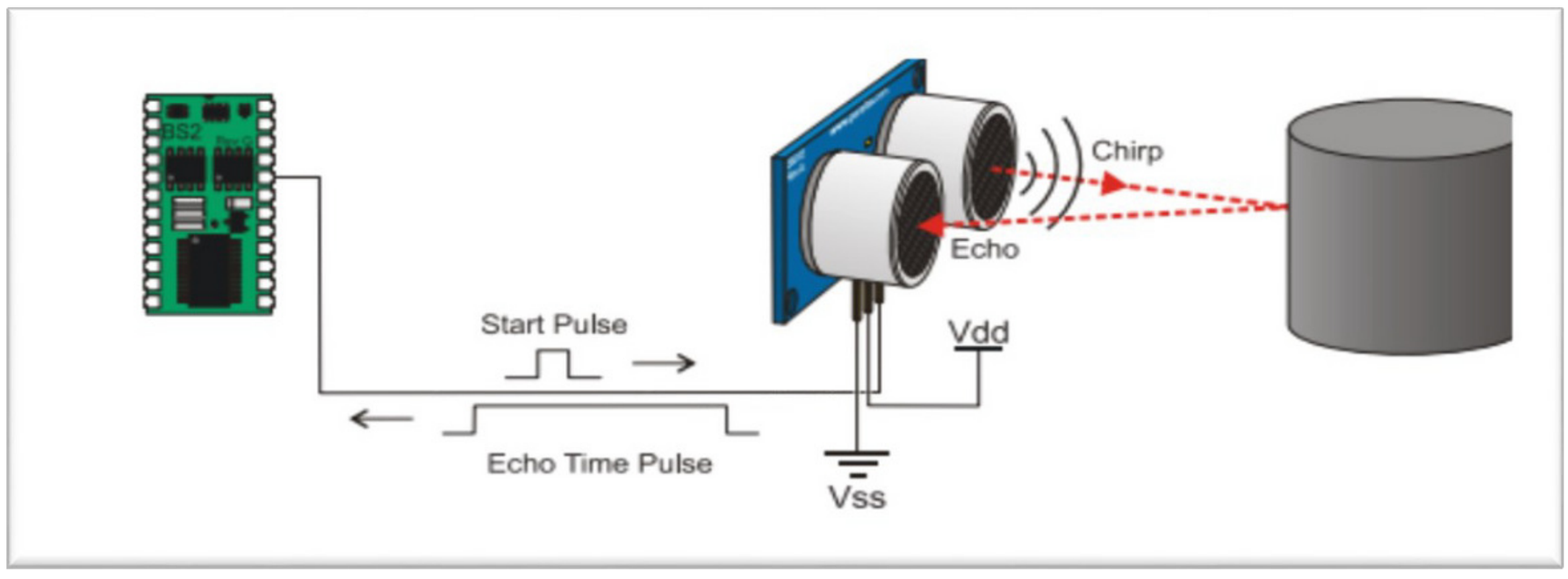


Figure 1.6: Ultrasonic Distance Sensor

Ultrasonic usage in our project:

We have four ultrasonic sensors in the car, located as follows: front, rear, right, and left. However, the right and left sensors are positioned towards the back and not in the middle, and this is due to a reason that we will mention below.

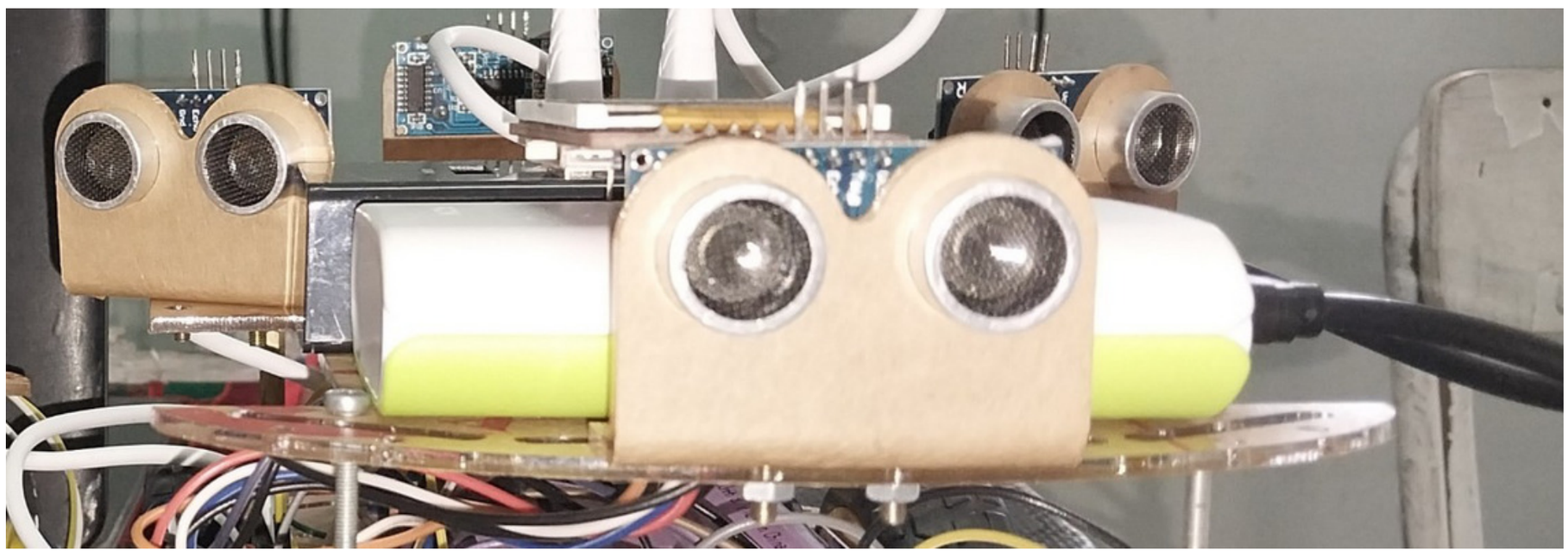


Figure 1.7: Ultrasonic sensor positions

The one located in the front is used to measure the distance between my car and the vehicle in front of me. Based on this distance, the driver is given a warning if the distance is not safe, or advised to reduce speed to avoid a collision. The ones located on the right and left are responsible for detecting blind spots, which is why they are positioned towards the back and not in the middle. This allows them to detect the presence of a vehicle in this area and determine the level of danger based on the distance. If a vehicle is detected in this area, the driver is advised not to move in that direction, and otherwise, they are given a warning only about the distance. The one located in the rear is used to measure the distance when parking or reversing. If an object is detected to be too close to the car, it will trigger a stop to prevent a collision. Otherwise, a notification about the distance will be displayed to the driver.

1.3 Subsystem Cases:

In our project, we deal with cases such as blind spots, forward direction, and parking the car. The handling of these issues is done only by giving warnings to the driver, and the car is not controlled except in special cases, such as stopping when parking if a collision is imminent. Also, ignoring the order in the case of blind spots is possible if it is confirmed that there is a vehicle or obstacle, or by reducing speed when there is a possibility of a collision with the vehicle in front when moving forward. In addition, the appropriate warning that expresses the situation will be displayed, such as the "unsafe distance" warning. Such situations are recognized using a distance measuring sensor and cameras. Also, through Wi-Fi to receive data from nearby cars.

When dealing with situations such as blind spots, forward direction, and parking the car, there are several common mistakes and pitfalls that should be avoided:-

1. Overreliance on technology: While technology can be beneficial in detecting blind spots or determining the direction of movement, it's important to remember that it's not foolproof. Drivers should always remain vigilant and aware of their surroundings, even when using advanced technology.
2. Lack of maintenance: Technology such as sensors and cameras require regular maintenance to function correctly. Failure to maintain these systems can lead to inaccurate readings, which can result in dangerous situations.
3. Poor calibration: Similarly, if the technology is not calibrated correctly, it can lead to inaccurate readings. It's important to ensure that the technology is calibrated correctly and regularly to avoid errors.
4. Neglecting to check blind spots manually: Even with advanced technology, it's important to manually check blind spots before changing lanes or making a turn to ensure that there are no other vehicles or obstacles in the way.
5. Ignoring warning signals: If the technology detects a potential danger, it will usually issue a warning signal. It's important not to ignore these signals and take appropriate action to avoid a collision or other dangerous situation.
6. Poor judgment: Finally, even with advanced technology, it's important to exercise good judgment when driving. This means understanding the limitations of the technology and making smart decisions based on the situation at hand.

1.3.1 blind spots:

The blind spot is an area around a vehicle that the driver cannot see directly through the side or rearview mirrors or by turning their head. Blind spots can vary depending on the vehicle's model, size, and design, and they can be a potential safety hazard for the driver and other road users.

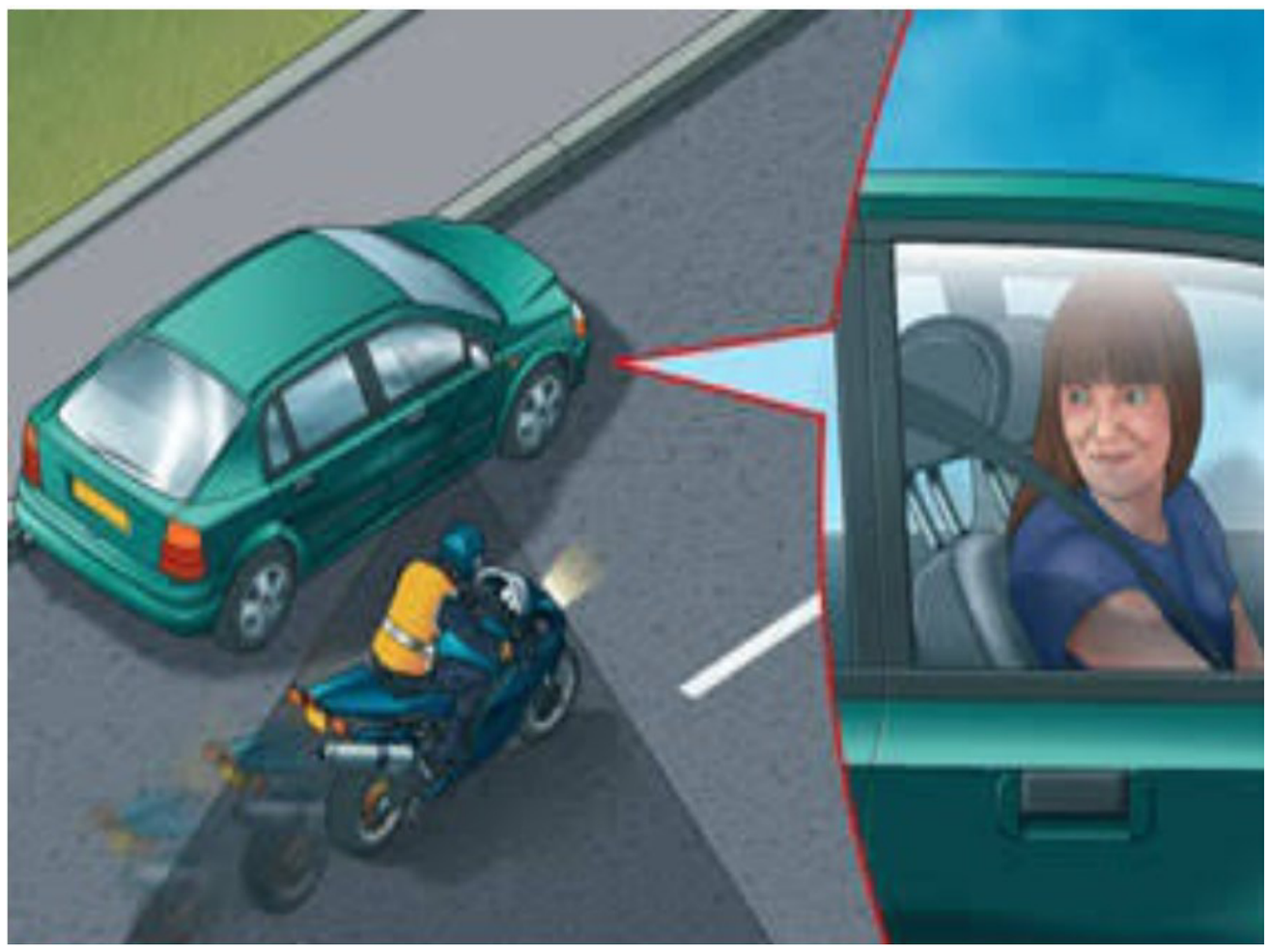


Figure 1.8: Blind spot case

Blind spots in a vehicle are usually located behind the driver's seat, on both sides of the vehicle, and sometimes in front of the vehicle. They can make it difficult for the driver to detect other vehicles, pedestrians, or objects in their path, especially when changing lanes, merging, or backing up. To mitigate these risks, many modern vehicles are equipped with technologies such as blind spot monitoring systems, which use sensors or cameras to detect objects in the blind spot and alert the driver with visual or auditory warnings. However, it is still important for drivers to be aware of their vehicle's blind spots and to check them manually before making any maneuvers.

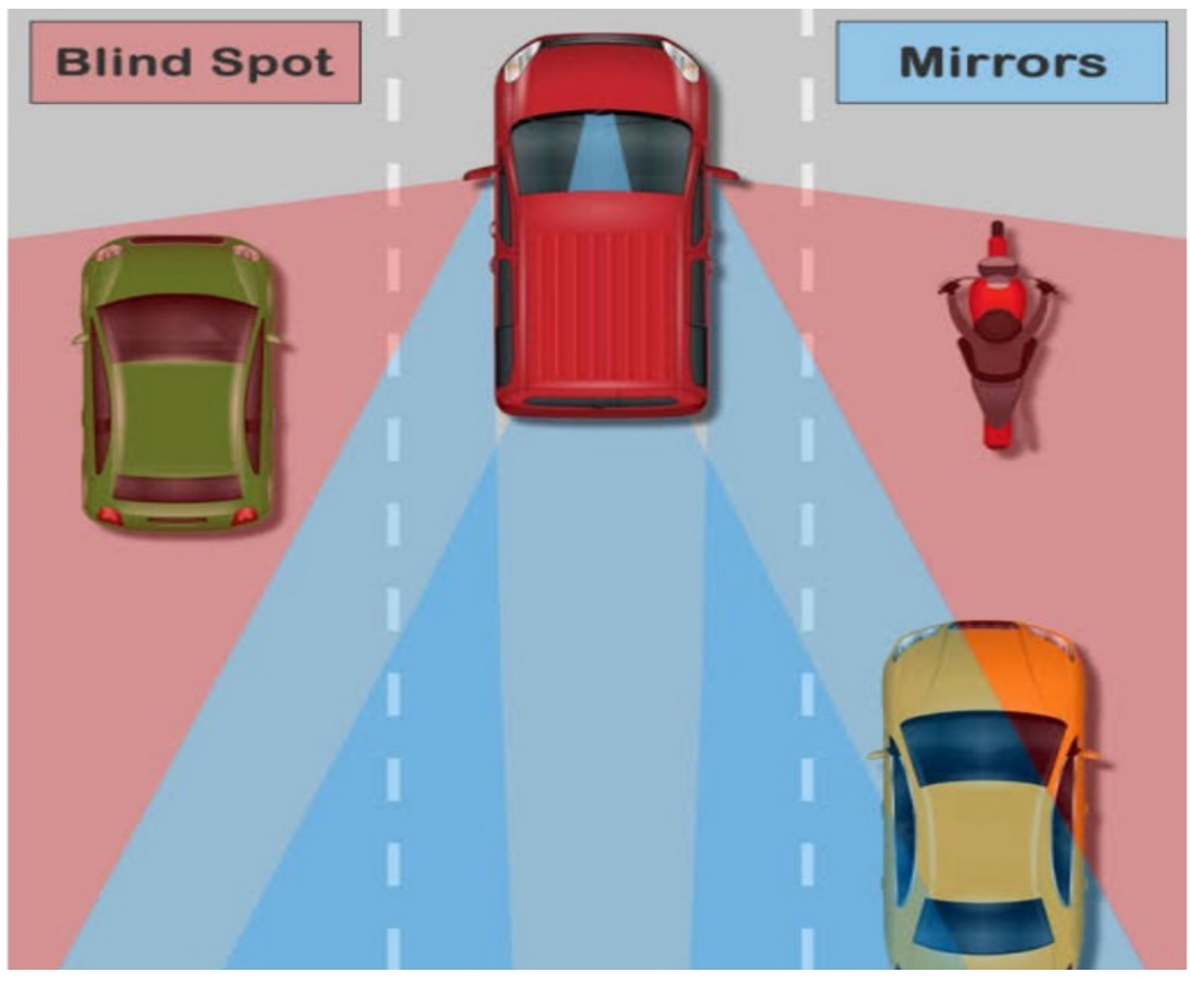


Figure 1.9: Blind Spot problem

How to determine the blind spot in our project?

It is determined by using the ultrasonic sensor located on the right side when turning right or on the left side when turning left.

Figure 1.10: detect blind spot using ultrasonic sensor

How to deal with the blind spot when it exists in the project?

When the driver commands the vehicle to move towards the right, for example

1. If the ultrasonic sensor reading is sufficiently large, the vehicle will move in that direction without giving any warning or changing the command.
2. If the ultrasonic sensor reading is small but sufficient for the vehicle to move, an alert will be activated to alert the driver to be cautious, and a notification 'Unsafe Distance' will be displayed on the screen in front of the driver.
3. If the ultrasonic sensor reading is small and insufficient, this will result in ignoring the command, and the vehicle will move forward without executing the command. An audible warning will be given to the driver, and a notification 'Command Ignored' will be displayed on the screen.

In all previous cases, the camera moves to accommodate the movement and displays the road on the screen in front of the driver to assist them in making the necessary decision.

Figure 1.11: displaying the blind spot to the driver

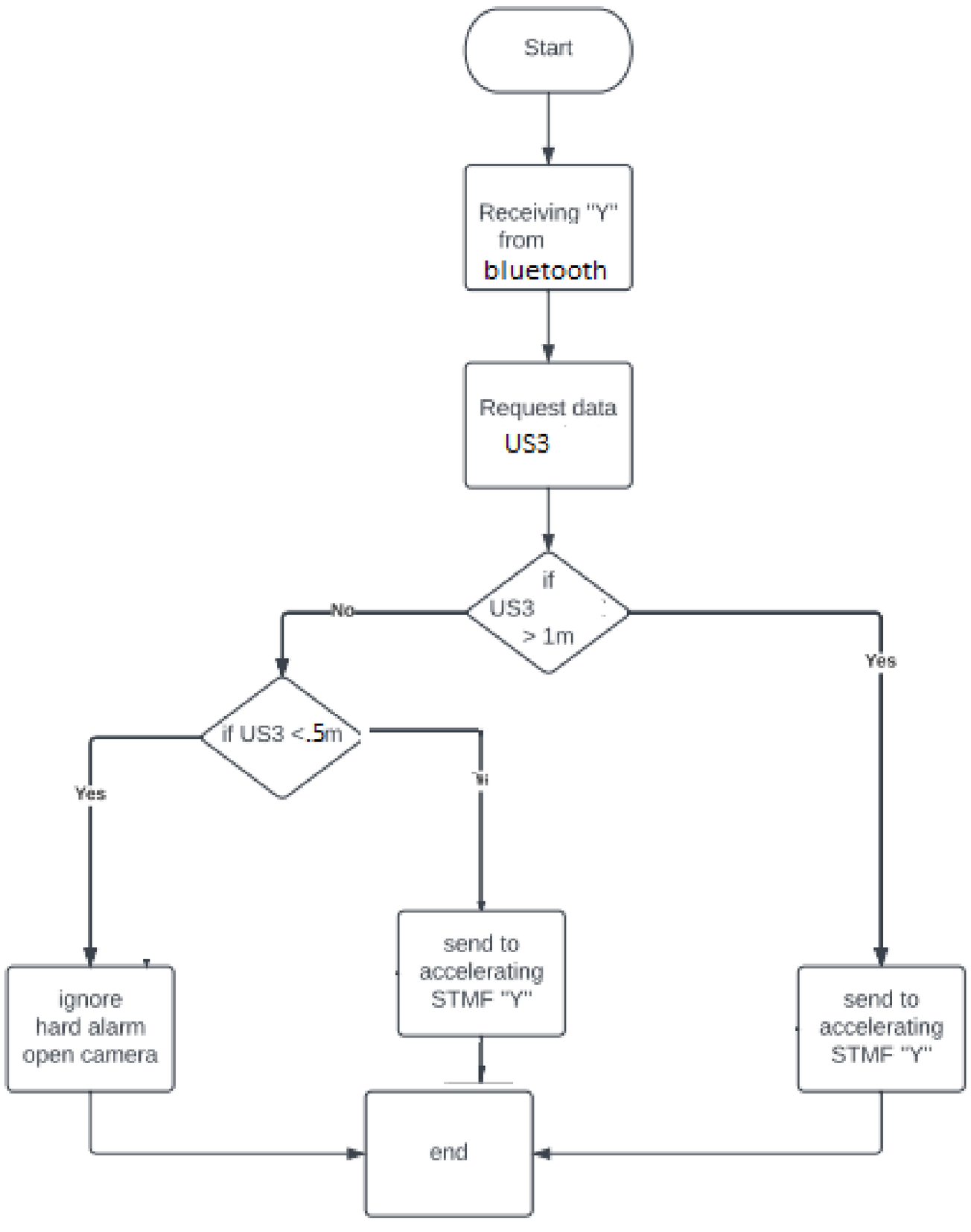


Figure 1.12: Blind spot Flowchart

1.3.2 forward moving:

If the driver is instructed to move forward here specifically, we do not rely solely on the ultrasonic sensors. Instead, we also request additional information from the Raspberry Pi to assist in making the right decision and to reduce collisions, driving on the road in a safer manner.
This case is the most complicated because it requires a lot of information, including the speed of the forward car and its comparison with its speed, as well as reading the distance sensor.

Based on the received data, there are many possibilities. Firstly, is there a car in front or is the road empty? This is identified through the Raspberry Pi, which determines whether there is a car in front or not, using both a camera and Wi-Fi. If there is a car, its speed is sent, and if there isn't, zero is sent.
After that, the distance sensor is read, and there are three possibilities: a safe distance that allows movement, an unsafe distance that still allows movement, and a short distance that does not allow movement. In the case of a car in front, the safe distance is between 10 to 15 meters, and it increases by 5 meters for every 10 km/h on highways.

Figure 1.13: safe distance with forward direction

The first scenario involves a car in front, followed by three possible cases. 1. The first case is a safe distance, and the direction and speed will be sent without change.

- **The first case** is a safe distance, and the direction and speed will be sent without change.

Figure 1.14: forward direction

- **In the second case**, the distance is not safe, meaning that the possibility of collision exists, but it is low. Therefore, a comparison is made between the speed of the car and the speed of the front car. If the speed of the car is less than or equal to the front car, the direction and speed will be sent without change, and a warning will be displayed on the screen that the distance is not safe. However, if the speed of the car is greater than the speed of the front car, the speed will be reduced to be equal to the speed of the front car to avoid collision. 3. In the third case, the distance is very short, so a loud warning is given to the driver with a notification that the distance is too short. The speed will be reduced to be equal to the speed of the front car.

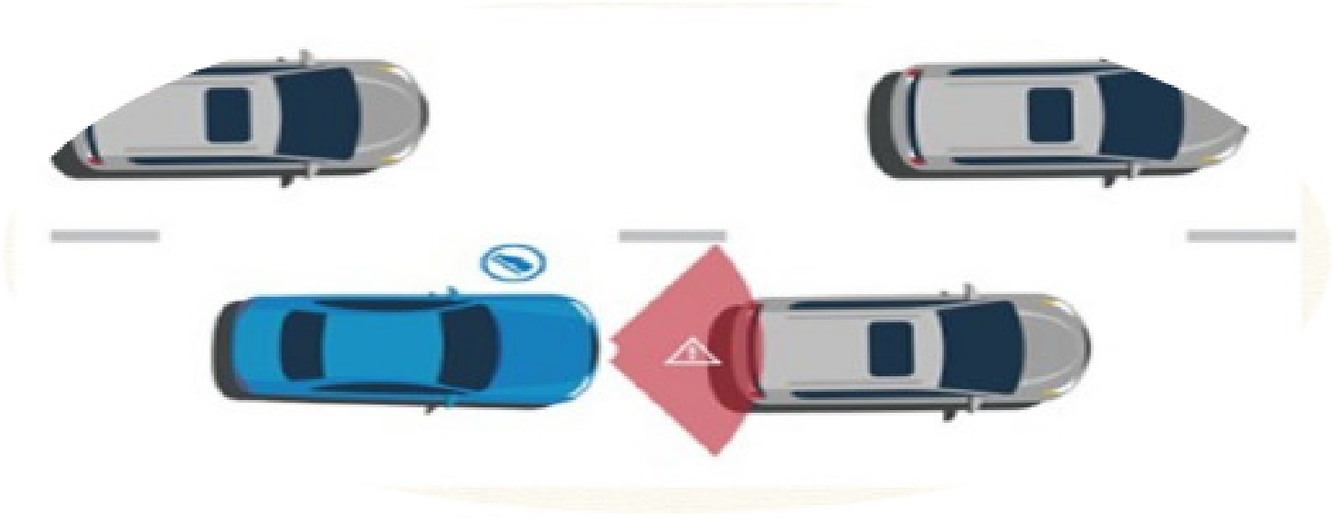


Figure 1.15: Forward direction with unsafe distance

In the second scenario, the Raspberry Pi has sent zero, meaning that there is no car in front. Therefore, the distance sensor reading in this case indicates the presence of an obstacle or object on the road. A loud warning is given to the driver with a notification of the situation.

Figure 1.16: Forward direction with obstacle

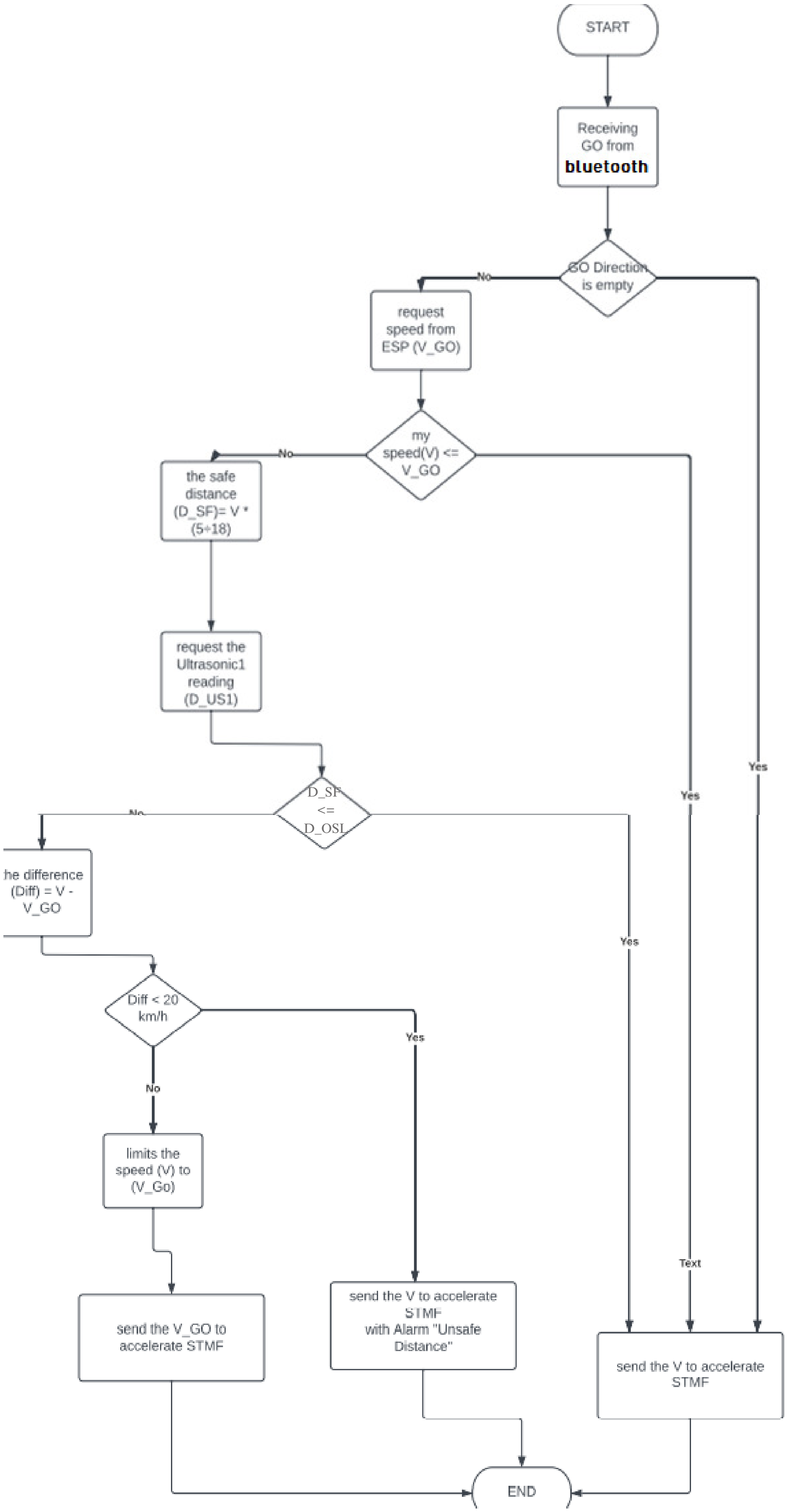


Figure 1.17: flow chart of GO direction

1.3.3 Backward moving:

This case pertains to reversing a car backwards, whether for parking or for other purposes.

Figure 1.18: Backward moving

In this case, and only at slow speeds, we can intervene in controlling the car as any damage caused will not result in further damage as in other cases. Therefore, if it is confirmed that there is an obstacle behind the car or an object at a very short distance, an immediate stop of the car should be made.

Type of parking in modern cars:

1. Autonomous Parking:

Autonomous parking is a feature found in some modern cars that allows the car to park itself without any intervention from the driver. The car uses various sensors and cameras to detect its surroundings and navigate into a parking space. The driver only needs to activate the feature and control the speed of the car, while the car takes care of the rest.

The autonomous parking feature can be a great help for drivers who struggle with parallel parking or parking in tight spaces.

Autonomous parking systems typically use a combination of cameras, radar, and ultrasonic sensors to detect the car's surroundings. These sensors generate a 360-degree view of the car's environment, which is then analyzed by the car's onboard computer to determine the best course of action for parking. The car will then steer itself into the parking space, while the driver controls the speed and monitors the process. Some autonomous parking systems can also detect and avoid obstacles while parking, and can even park the car in a perpendicular or diagonal parking space.

This feature can be especially helpful for drivers who have difficulty judging the distance between their car and other objects while parking.

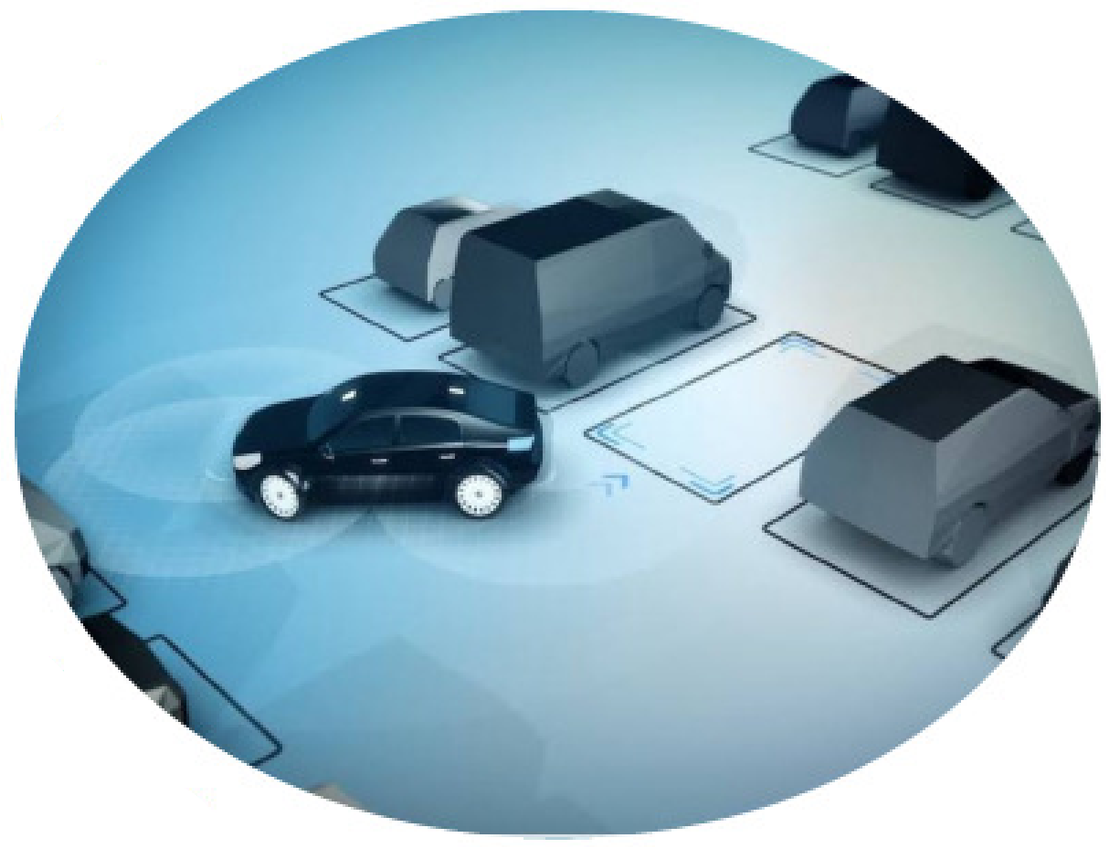


Figure 1.19: Autonomous Parking

2. Parking Assistant:-

Parking assistant is another feature found in modern cars that provides an alarm to the driver when the car gets too close to an obstacle while parking. This system uses sensors and cameras to detect the distance between the car and any objects in its surroundings. When the car gets too close to an obstacle, the system will beep or sound an alarm to alert the driver. The parking assistant feature is designed to help drivers avoid collisions while parking and can be especially useful in tight parking spaces or crowded areas.

Figure 1.20: Assistance Parking

It's worth noting that while both autonomous parking and parking assistant features can be helpful tools for drivers, they are not a substitute for safe driving practices. Drivers should always remain alert and aware of their surroundings while parking, and should never rely solely on these features to park their car safely. Overall, both autonomous parking and parking assistant features can be helpful tools for drivers when parking their cars. While autonomous parking takes care of the entire parking process, parking assistant provides an additional level of safety by alerting the driver to potential hazards. In our project, we worked on the idea of a parking assistant system. It is operated through a distance sensor and a camera, where the sensor measures the distance and the camera displays the cars or objects behind the vehicle. Based on reading the ultrasonic sensor, there are three possibilities that can occur.

1. If the ultrasonic sensor reading is sufficiently large, the vehicle will move in that direction without giving any warning or changing the command.
2. If the ultrasonic sensor reading is small but sufficient for the vehicle to move, an alert will be activated to alert the driver to be cautious, and a notification 'Unsafe Distance' will be displayed on the screen in front of the driver.
3. If the ultrasonic sensor reading indicates a short and insufficient distance, an immediate stop of the vehicle will be activated. An audible warning will be given to the driver, and a notification “Car stop” will be displayed on the screen.

In all previous cases, the camera moves to the movement direction and displays the road on the screen in front of the driver to assist them in making the necessary decision.

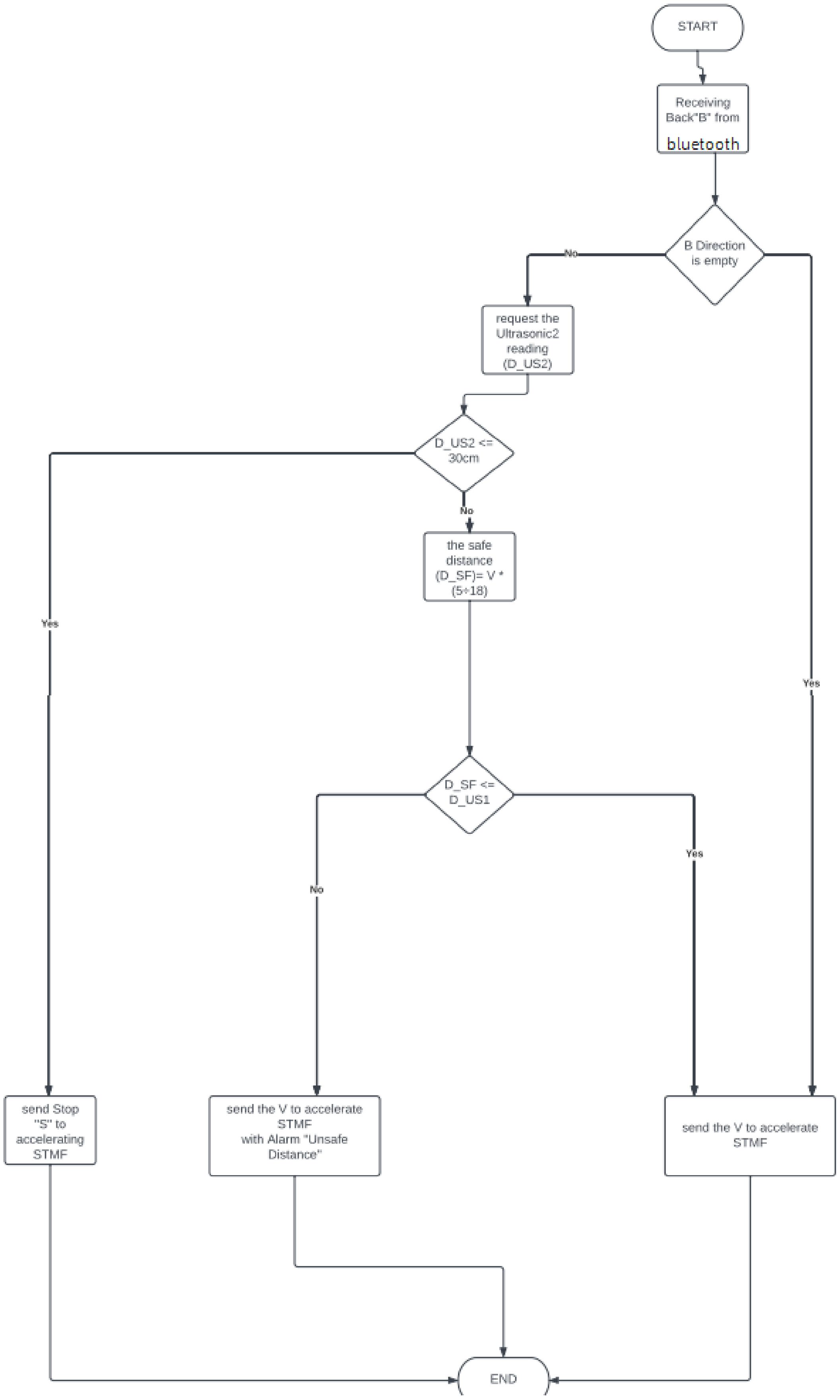


Figure 1.21: Flowchart of back case

1.3.4 Right & Left Case:

If the direction is changed to the right or left, the direction and speed are sent unchanged.

Figure 1.22: Right case

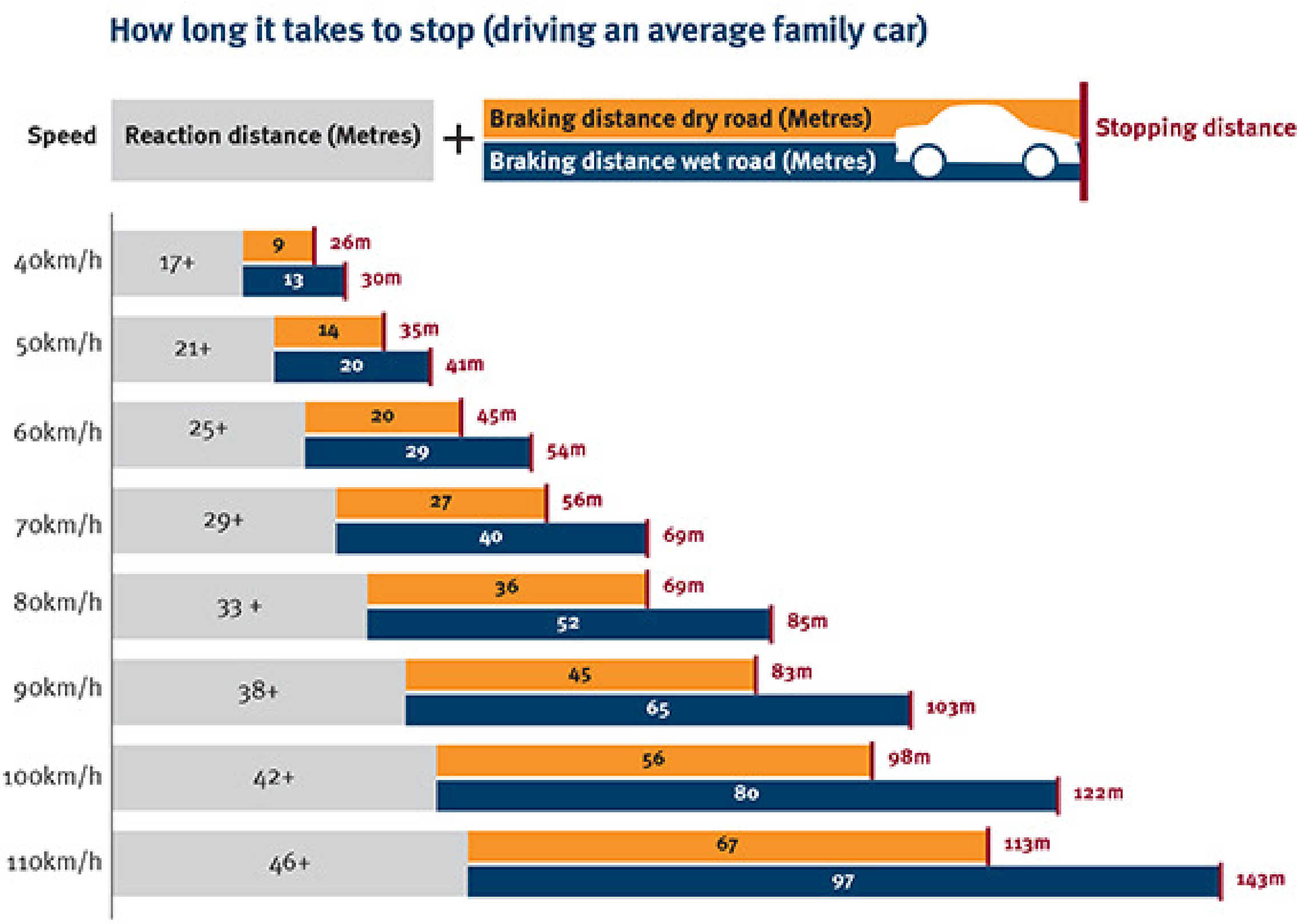


Table 1.1: safe distance between two cars

1.4 Raspberry pi Role in Vehicle-to Vehicle:

In the preceding chapters, we Explained what Embedded Linux Systems is and what is OpenCv , now it is time to build The application related to their meaning , so in this chapter we will Explain the device application.

1.4.1 Introduction:

The Device application is divided into four main Tasks:

- The Communication between the Embedded Linux development board and the STM.
- The STM will send data to the Embedded Linux development board to start the Camera and also will receive another data from raspberry pi.

1.4.2 Vehicle-to-Vehicle (V2V):

Vehicle-to-vehicle (V2V) communication enables vehicles to wirelessly exchange information about their speed, location, and heading. The technology behind V2V communication allows vehicles to broadcast and receive omni-directional messages (up to 10 times per second), creating a 360-degree “awareness” of other vehicles in proximity. Vehicles equipped with appropriate software (or safety applications) can use the messages from surrounding vehicles to determine potential crash threats as they develop. The technology can then employ visual, tactile, and audible alerts or, a combination of these alerts to warn drivers. These alerts allow drivers the ability to take action to avoid crashes [4].

V2V communication systems use wireless communication technologies such as Wi-Fi, Bluetooth, and cellular networks to allow vehicles to share data with each other.

The purpose of V2V communication is to improve road safety and traffic efficiency by enabling vehicles to "see" each other and react to potential hazards or traffic situations more quickly and effectively. For example, V2V communication can be used to help prevent collisions by alerting drivers to potential hazards, such as a car running a red light or a pedestrian crossing the street.

V2V communication is an important component of the broader concept of Vehicle-to-Everything (V2X) communication, which includes not only Vehicle-to-Vehicle communication but also Vehicle-to-Infrastructure (V2I), Vehicle-to-Pedestrian (V2P), and Vehicle-to-Cloud (V2C) communication. V2V communication is considered a key technology for the development of autonomous and connected vehicles, and has the potential to significantly enhance road safety and efficiency [4].

1.4.3 Raspberry Pi:

The Raspberry Pi is a credit-card-sized fully functioning computer (System on chip) created by the non-profit Raspberry Pi Foundation in the UK. It's considered as a SOC (system on chip) device that runs on a Linux OS specially designed for it, called Raspbian.

Raspbian is the official OS for Raspberry Pi, whereas other third-party OSes like Firefox OS, Android, RISC OS, and Ubuntu Mate. can be installed on Pi. Like a computer, it has memory, processor, USB ports, audio output, and a graphic driver for HDMI output [49].

The Raspberry Pi is amazing at two levels the advanced functionality that you get in a credit card-sized SBC (Single Board Computer) and its price. Even with today's Pi competitors, the Raspberry Pi reigns supreme because few can beat its price. Further, it enjoys great software and community support. I used Raspberry Pi 3B+ for The Project and its specifications are:

- SoC: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- GPU: Broadcom Videocore-IV
- RAM: 1GB LPDDR2 SDRAM
- Networking: Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
- Bluetooth: Bluetooth 4.2, Bluetooth Low Energy (BLE)
- Storage: Micro-SD
- GPIO: 40-pin GPIO header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- Dimensions: 82mm x 56mm x 19.5mm, 50g One of the most widely used programming languages on the Raspberry Pi is Python. Python is a high-level, general-purpose, very popular programming language, Interpreted, Platform Independent, Free, Open Source, and supports many libraries.

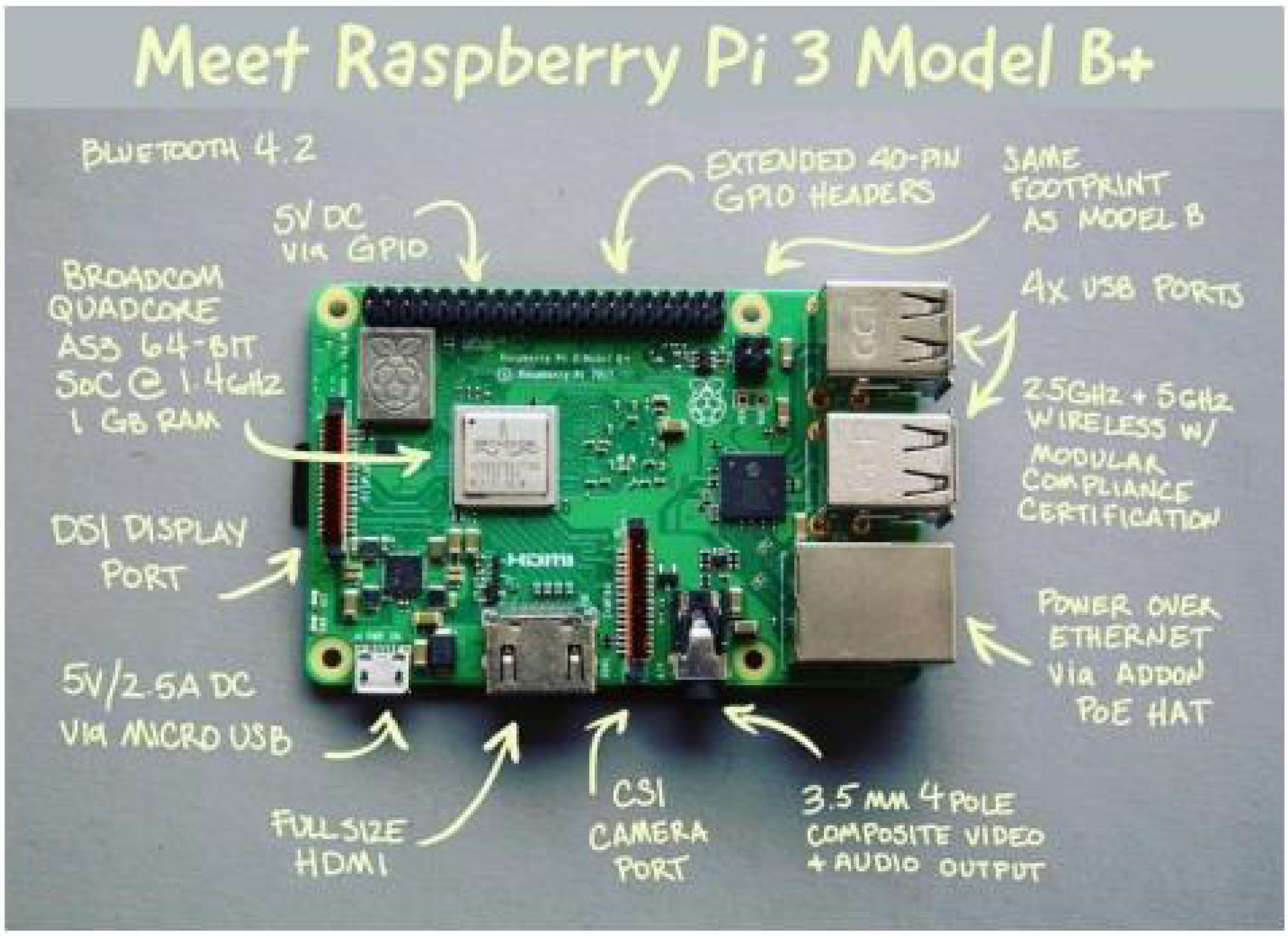

Meet Raspberry Pi 3 Model B+
BLUETOOTH 4.2
5V DC VIA GPIO
EXTENDED 40-PIN GPIO HEADERS
SAME FOOTPRINT AS MODEL B
BROADCOM QUADCORE ASS 64-BIT SoC @ 1.4GHz 1 GB RAM
4X USB PORTS
2.5GHz + 5GHz WIRELESS W/ MODULAR COMPLIANCE CERTIFICATION
DSI DISPLAY PORT
POWER OVER ETHERNET VIA ADDON PoE HAT
5V/2.5A DC VIA MICRO USB
FULL SIZE HDMI
CSI CAMERA PORT
3.5 MM 4 POLE COMPOSITE VIDEO + AUDIO OUTPUT

Figure 1.23: Raspberry Pi 3B+

1.4.4 Task 1:

The task : Communication between the Embedded Linux development board and the STM.

1.4.4.a Description:

There are several communication protocols that can be used to establish communication between an STM microcontroller and a Raspberry Pi, including:

- UART (Universal Asynchronous Receiver/Transmitter): UART is a simple serial communication protocol that is commonly used for short-range communication between microcontrollers and computers. As mentioned earlier, UART can be used to establish communication between an STM microcontroller and a Raspberry Pi by connecting the UART pins of both devices.
- I2C (Inter-Integrated Circuit): I2C is a serial communication protocol that is commonly used for short-range communication between integrated circuits. I2C uses a master-slave architecture, where the Raspberry Pi acts as the master and the STM microcontroller acts as the slave. I2C requires only two wires for communication, making it a popular choice for applications with limited wiring.
- SPI (Serial Peripheral Interface): SPI is a synchronous serial communication protocol that is commonly used for short-range communication between microcontrollers and peripherals such as sensors and displays. SPI requires four wires for communication: a clock signal, a data signal, a slave select signal, and a ground reference.

- CAN (Controller Area Network): CAN is a serial communication protocol that is commonly used for communication between automotive components and other industrial systems. CAN is a differential signaling protocol, which means that it uses two wires for communication and is resistant to noise and interference.

We choice UART communication protocol between an STM microcontroller and a Raspberry Pi based on the specific requirements of the application, including the distance between the two devices, the amount of data that needs to be transmitted, and the noise and interference in the environment.

1.4.4.b Hardware Components:

Three wires between raspberry pi and STM , one for TX, the second for RX, and the third for ground.

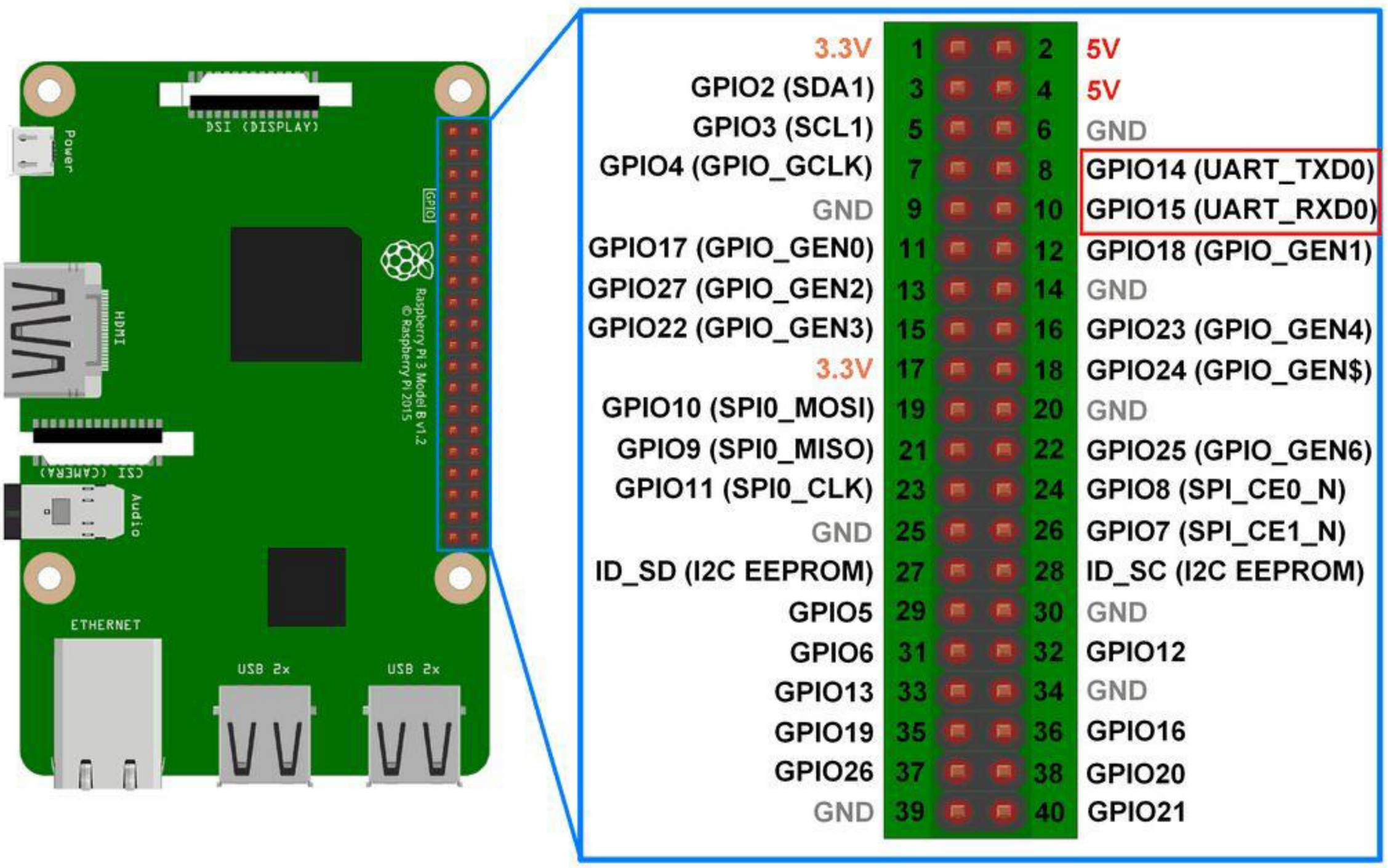

DSI (DISPLAY)
Power
GPIO
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
Raspberry Pi 3 Model B v1.2
© Raspberry Pi 2015
3.3V 1 2 5V
GPIO2 (SDA1) 3 4 5V
GPIO3 (SCL1) 5 6 GND
GPIO4 (GPIO_GCLK) 7 8 GPIO14 (UART_TXD0)
GND 9 10 GPIO15 (UART_RXD0)
GPIO17 (GPIO_GEN0) 11 12 GPIO18 (GPIO_GEN1)
GPIO27 (GPIO_GEN2) 13 14 GND
GPIO22 (GPIO_GEN3) 15 16 GPIO23 (GPIO_GEN4)
3.3V 17 18 GPIO24 (GPIO_GEN$)
GPIO10 (SPI0_MOSI) 19 20 GND
GPIO9 (SPI0_MISO) 21 22 GPIO25 (GPIO_GEN6)
GPIO11 (SPI0_CLK) 23 24 GPIO8 (SPI_CE0_N)
GND 25 26 GPIO7 (SPI_CE1_N)
ID_SD (I2C EEPROM) 27 28 ID_SC (I2C EEPROM)
GPIO5 29 30 GND
GPIO6 31 32 GPIO12
GPIO13 33 34 GND
GPIO19 35 36 GPIO16
GPIO26 37 38 GPIO20
GND 39 40 GPIO21

Figure 1.24: raspberry pi UART

1.4.5 Task 2:

The task : The STM will send data to the Embedded Linux development board to start the Camera and also will receive another data from raspberry pi.

1.4.5.a Description:

STM will send directions to raspberry pi :

- Go
- Back
- Backward left
- Backward right
- Forward left
- Forward right

The raspberry pi in my car will receive color and speeds 0,1,2 and 3 from the raspberry pi in the other car by using WiFi and socket programming .

If the Raspberry pi received go :the camera will be open in the forward direction and detect the color of the other car by using opencv and compare the detection color with the color that came from the Raspberry pi in the other car , if they are similar ,this color will be sent to the STM and also speeds 1,2,3 of the other car will be sent to STM, and if they are not similar speed 0 will be sent to STM.

If the received data is back : The stepper motor connected to the camera on the Raspberry pi will rotate 180 degree and the camera will be open .

1.4.5.b Blind Spot When Driving a Car On The Road:

A blind spot is a point around the car where the driver cannot detect the presence of objects or other vehicles. In other words, if there are other vehicles coming from the area, it becomes more prone to collisions due to the limited visibility of these vehicles. Blind spots can be a significant safety concern, as they can make it difficult for drivers to see other vehicles, pedestrians, or obstacles that may be in their path. Blind spots are typically located on the sides and rear of the vehicle, and can vary in size depending on the make and model of the vehicle. They are particularly common in larger vehicles such as trucks and SUVs, which have larger blind spots due to their size and shape. To help address the issue of blind spots, many newer vehicles are equipped with blind spot monitoring systems. These systems
use sensors and cameras to detect other vehicles in the blind spot and provide visual or audio alerts to the driver. Some systems can even take corrective action, such as applying the brakes or steering the vehicle back into its lane, to help prevent collisions.

In addition to blind spot monitoring systems, there are also several strategies that drivers can use to help minimize the risk of blind spot-related accidents. These include adjusting the side mirrors to provide a wider field of view, physically looking over the shoulder before changing lanes or turning, and avoiding driving in other drivers' blind spots whenever possible.

We will help the driver by opening the camera in some directions to avoid the blind spot area.

- If the receive data is backward left : The stepper motor connected to the camera on the Raspberry pi will rotate 90 degree anti-clockwise and the camera will be open .
- If the receive data is backward right : The stepper motor connected to the camera on the Raspberry pi will rotate 90 degree clockwise and the camera will be open.
- If the receive data is forward left : The stepper motor connected to the camera on the Raspberry pi will rotate 45 degree anti-clockwise and the camera will be open .
- If the receive data is forward right : The stepper motor connected to the camera on the Raspberry pi will rotate 45 degree clockwise and the camera will be open.

Acceleration Subsystem

2

In this chapter, we'll talk about acceleration subsystem with it's functionality, hardware components used in this subsystem, subsystem software to control of it's functionality, also we will discuss the controllers used.

2.1 Driving Mobile Vehicle

For main vehicle:

At first, vehicle take the initial speed (zero) and initial direction (stop) until received data from Processing STM using UART communication protocol.

Acceleration STM received data from Processing STM, this data contains speed, direction and flags to indicate any un safed case.

- Directions received: (Go, Back, Right, Left, Forward Right, Forward Left, Backward Right, Backward Left).
- Speed recieved: (Speed1, Speed2, Speed3).
- Flags received: (Flag 0 for indicate no error and safety case, Flag1 for indicate un safed distance, Flag 2 for indicate limitation speed or ignore order, Flag 3 to forced car to be must stop.

Then display speed, direction and indicated messages on the TFT and send this data to RPI.

For assistance vehicle:

At first, vehicle take the initial speed (zero) and initial direction (stop) until received data from bluetooth module.

Acceleration STM received data from bluetooth module, this data contains speed and direction.

- Directions received: (Go, Back, Right, Left, Forward Right, Forward Left, Backward Right, Backward Left).
- Speed recieved: (Speed1, Speed2, Speed3).

Then display speed and direction on the LCD and send this data to RPI.

2.1.1 Hardware Component and Controllers Used:

As mentioned, with the development of vehicle comes the development of the tools used. Therefore, hardware technology must keep up with this development. Robot hardware can be divided into three categories:

- **Controllers**: In the past, vehicles were controlled by large and expensive computer systems that could not be carried and had to be linked via cable or wireless modems. Nowadays, we can build small vehicles with numerous actuators and sensors that are controlled by inexpensive, small, and lightweight embedded computer systems that are carried on board the robot.
- **Actuators**: Controllers are the brain of robot to decide the proper action, sensors are the senses of the robot to deliver the environmental variables. So, we need devices to perform this action. There are several types of actuators and we'll be interested in DC motors that are used for driving mobile robots.
- **Power System**: All hardware components need power to work, also not all hardware work at the same power level. Therefore, we must have a good power system to deliver all desired power to all hardware components [4].

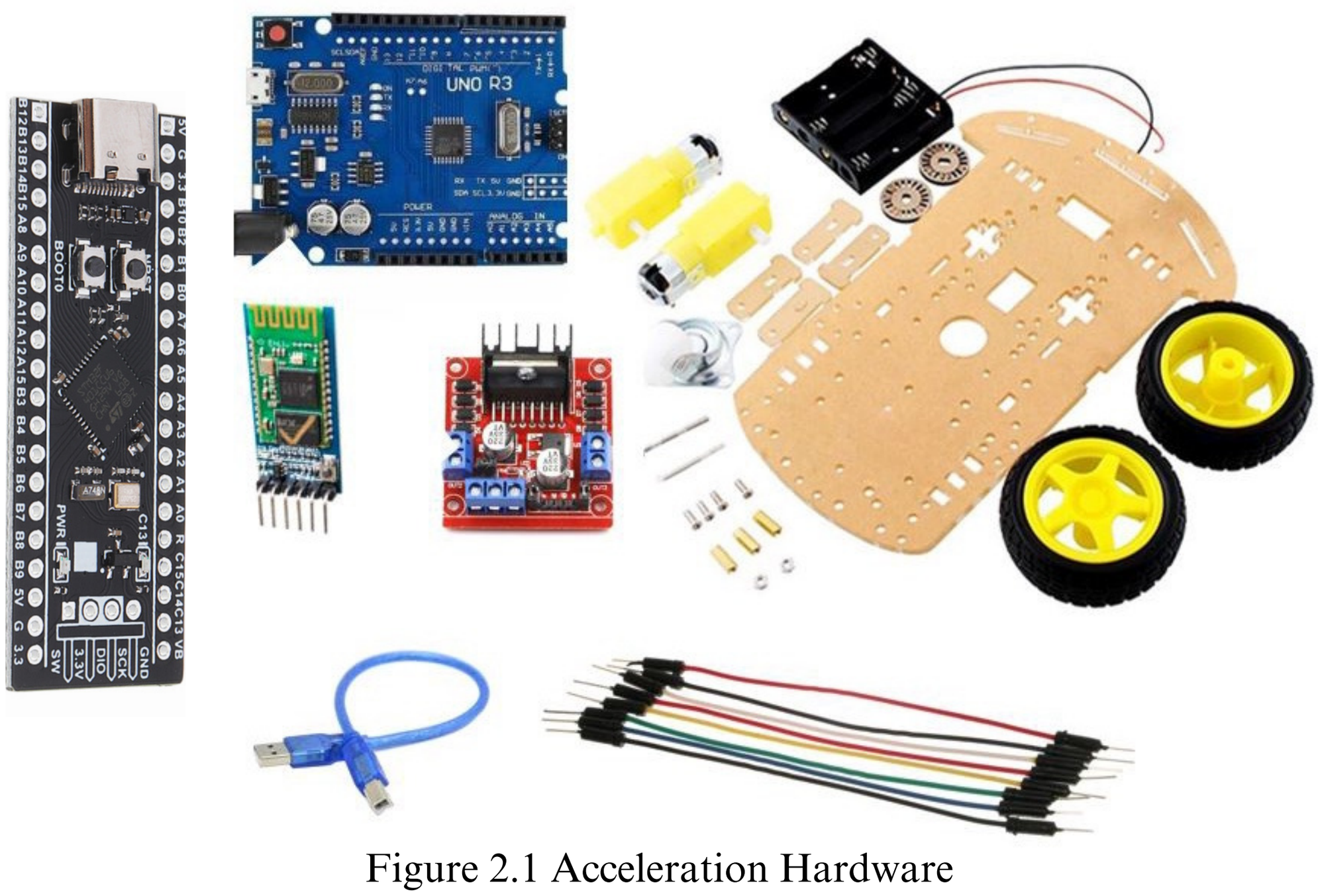


Figure 2.1 Acceleration Hardware

2.1.1.a Controllers:

We'll talk about two controllers (Arduino used at assistance vehicle and STM used at main vehicle):

- ***Arduino:*** The Arduino family of controllers was developed in 2005 by the interaction Design Institute Ivrea in Italy. Hardware and software are open source, which added to their success and the availability of low-cost versions. The Nano controller has a small footprint and is ideal for small projects as it only costs a few dollars. Larger versions such as the Arduino Uno and Arduino Mega can take add-on boards or shields, but for simple projects, the Nano is all we need. The Arduino, like many other controller boards such as the AVR butterfly, uses an Atmel AVR 8-bit microcontroller. For the Nano version, it is an Atmel ATmega328P. Programming can be easily done in C using Atmel studio or in a simplified C version using the Arduino app. The Arduino has a small built-in ROM memory and cannot be extended. Powering up is simply by connecting its USB socket to a power supply or a PC. Upload and download of programs and data can also be accomplished over the USB link. We'll focus on the Arduino UNO and its features.

Arduino UNO is an open-source development board based on Atmega328 AVR microcontroller. This microcontroller is an 8- bit microcontroller.
This board can be programmed using programmed using wiring/ processing language. It includes:

1. 14 digital I/O pins out of which 6 pins can be used as PWM outputs
2. 6 analog pins
3. A 16 MHz crystal oscillator
4. A USB connection
5. A power Jack
6. A reset buttons

-*The purpose of the major pins and ports:*

- Digital Input/ Output Pins: It is used to transmit and receive the digital signals respectively.
- PWM Outputs: PWM (Pulse Width Modulation) pin is used to control the signals. For example, controlling the speed of the motors, brightness of LED.
- Analog Pins: It performs the function of reading analog signals. For example, reading the sensors data. Also, it serves as general purpose input/output pins.
- USB: It serves two purposes i.e.; it can connect Mega 2560 board to your PC to program your board as well as it can be used to dram power to the board.
- Power Jack: It is used to supply power to the board externally.

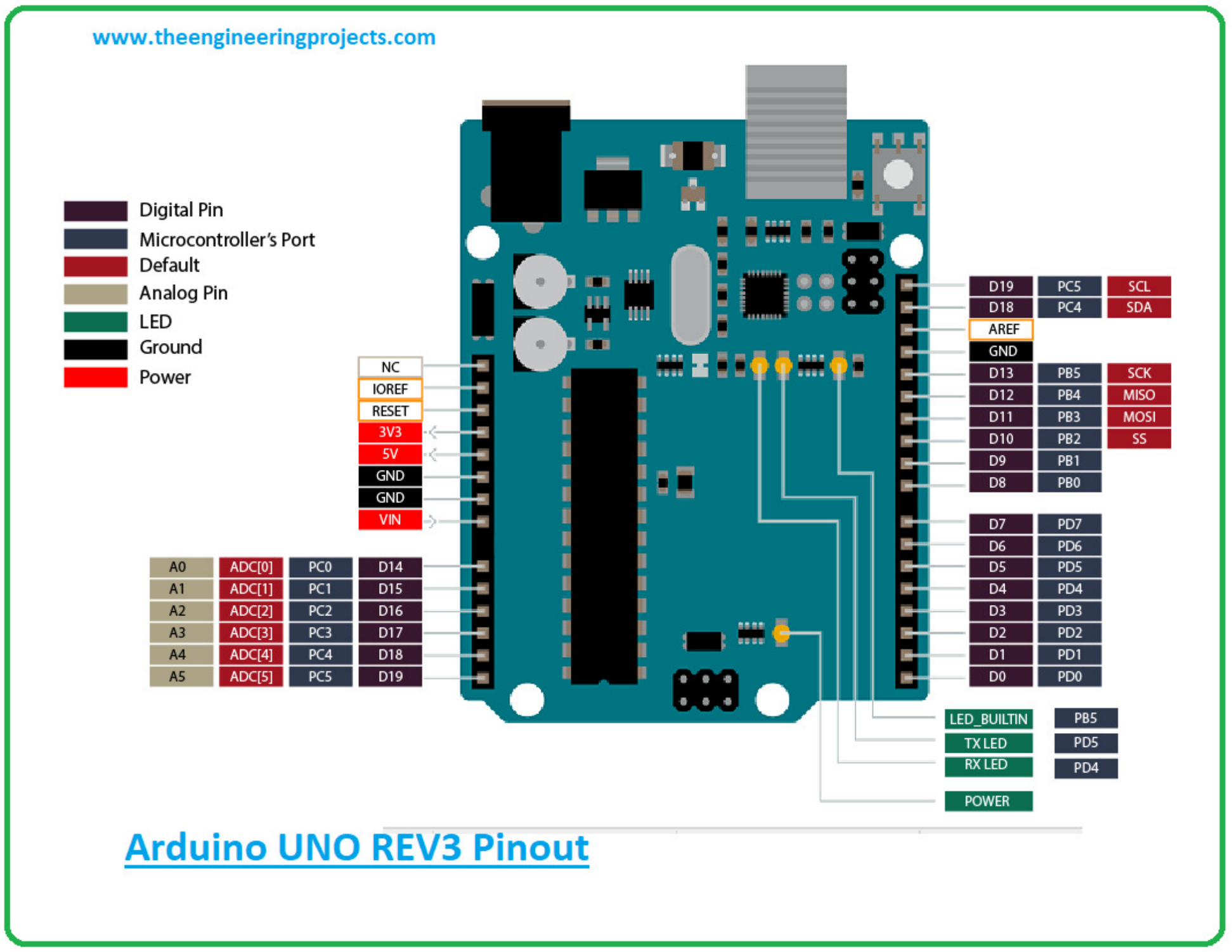


Figure 2.2 arduino uno pinout

Microcontroller	ATmega328P
Operating Volt.	5V
Input Volt. (recommended)	7-12V
Input Volt. (limit)	6-20V
Digital I/O Pins	14 (6 PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm

table 2.1 Technical Features arduino uno

- ***STM32F401CCU6:*** The STM32F401xB/STM32F401xC devices are based on the high-performance Arm® Cortex® -M4 32-bit RISC core operating at a frequency of up to 84 MHz. The Cortex®-M4 core features a Floating point unit (FPU) single precision which supports all Arm single-precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security.

The STM32F401xB/STM32F401xC incorporate high-speed embedded memories (up to 256 Kbytes of Flash memory, up to 64 Kbytes of SRAM), and an extensive range of enhanced I/Os and peripherals connected to two APB buses, two AHB buses and a 32-bit multi-AHB bus matrix

All devices offer one 12-bit ADC, a low-power RTC, six general-purpose 16-bit timers including one PWM timer for motor control, two general-purpose 32-bit timers. They also feature standard and advanced communication interfaces.

The STM32F401xB/STM32F401xC operate in the - 40 to + 125 °C temperature range from a 1.7 (PDR OFF) to 3.6 V power supply. A comprehensive set of power-saving mode allows the design of low-power applications.

These features make the STM32F401xB/STM32F401xC microcontrollers suitable for a wide range of applications.

- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 84 MHz, memory protection unit, 105 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - Up to 256 Kbytes of Flash memory
  - 512 bytes of OTP memory
  - Up to 64 Kbytes of SRAM
- CRC calculation unit
- 96-bit unique ID
- RTC: sub second accuracy, hardware calendar
- All packages are ECOPACK2
- Up to 11 communication interfaces
- 1×12-bit, 2.4 MSPS A/D converter: up to 16 channels
- General-purpose DMA: 16-stream DMA controllers with FIFOs and burst support
- Up to 11 timers: up to six 16-bit, two 32-bit timers up to 84 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input, two watchdog timers (independent and window) and a SysTick timer.

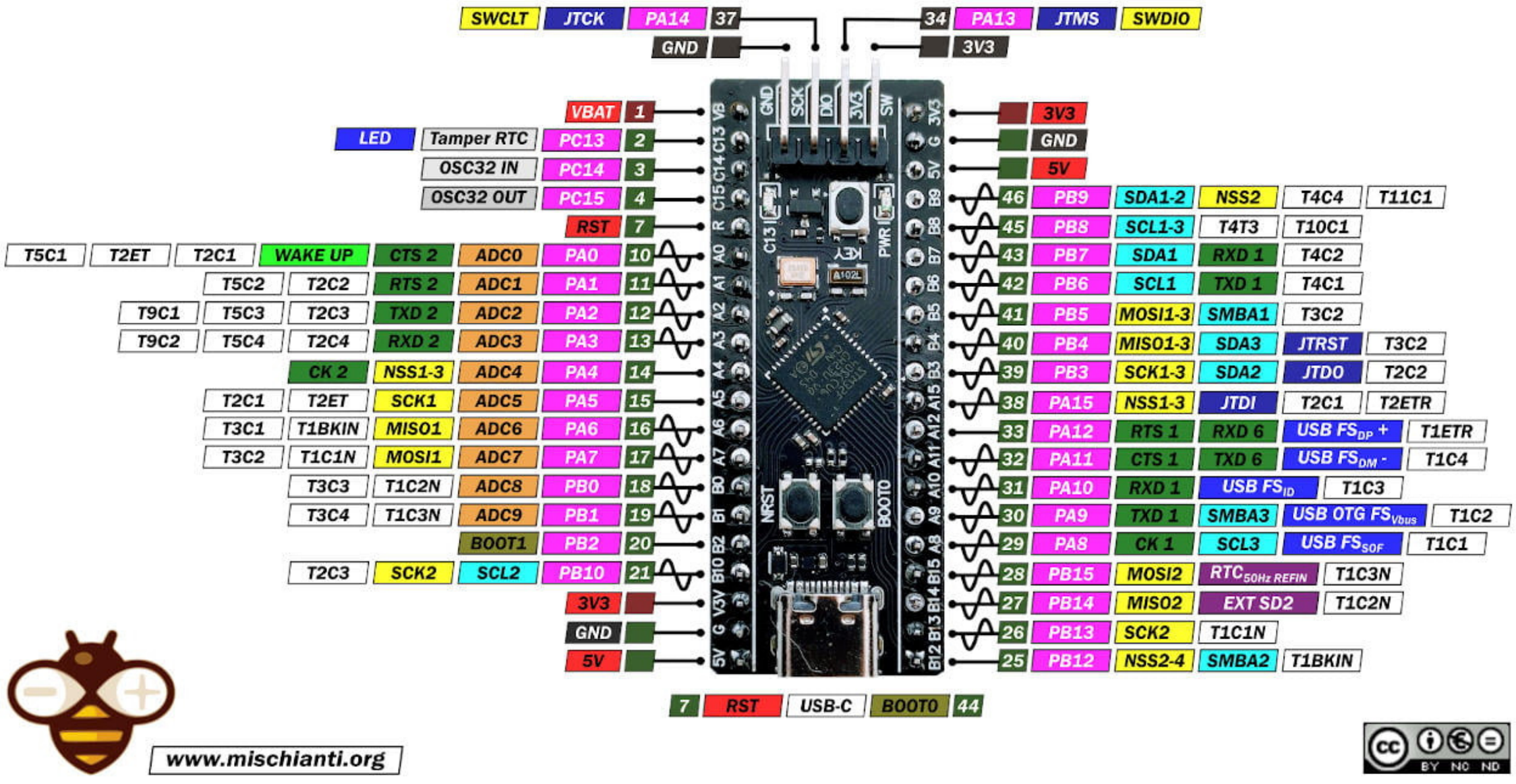

WeAct STM32F401
PINOUT
SWCLT JTCK PA14 37
GND
34 PA13 JTMS SWDIO
3V3
VBAT 1
LED Tamper RTC PC13 2
OSC32 IN PC14 3
OSC32 OUT PC15 4
RST 7
T5C1 T2ET T2C1 WAKE UP CTS 2 ADC0 PA0 10
T5C2 T2C2 RTS 2 ADC1 PA1 11
T9C1 T5C3 T2C3 TXD 2 ADC2 PA2 12
T9C2 T5C4 T2C4 RXD 2 ADC3 PA3 13
CK 2 NSS1-3 ADC4 PA4 14
T2C1 T2ET SCK1 ADC5 PA5 15
T3C1 T1BKIN MISO1 ADC6 PA6 16
T3C2 T1C1N MOSI1 ADC7 PA7 17
T3C3 T1C2N ADC8 PB0 18
T3C4 T1C3N ADC9 PB1 19
BOOT1 PB2 20
T2C3 SCK2 SCL2 PB10 21
3V3
GND
5V
3V3
GND
5V
46 PB9 SDA1-2 NSS2 T4C4 T11C1
45 PB8 SCL1-3 T4T3 T10C1
43 PB7 SDA1 RXD 1 T4C2
42 PB6 SCL1 TXD 1 T4C1
41 PB5 MOSI1-3 SMBA1 T3C2
40 PB4 MISO1-3 SDA3 JTRST T3C2
39 PB3 SCK1-3 SDA2 JTDO T2C2
38 PA15 NSS1-3 JTDI T2C1 T2ETR
33 PA12 RTS 1 RXD 6 USB FSDP + T1ETR
32 PA11 CTS 1 TXD 6 USB FSDM - T1C4
31 PA10 RXD 1 USB FSID T1C3
30 PA9 TXD 1 SMBA3 USB OTG FSVbus T1C2
29 PA8 CK 1 SCL3 USB FSSOF T1C1
28 PB15 MOSI2 RTC50Hz REFIN T1C3N
27 PB14 MISO2 EXT SD2 T1C2N
26 PB13 SCK2 T1C1N
25 PB12 NSS2-4 SMBA2 T1BKIN
7 RST USB-C BOOT0 44
www.mischianti.org

Figure 2.3 Pinout of STM32F401

2.1.1.b Actuators:

There are many ways that robotic actuators can be built. Most prominently, these are electric motors or pneumatic actuators with electric valves. We will concentrate on electric actuators using Direct Current (DC).

Single-phase DC electric motors are arguably the most used method for locomotion in mobile robots. DC motors are clean, quiet, and can produce sufficient power for a variety of tasks. They are much easier to control than pneumatic actuators, which are frequently used when high torques are required.

The first step when building robot hardware is to select the appropriate motor system. The best choice is an encapsulated motor combination comprising:

- DC motor
- Gearbox
- Optical or magnetic encoder (dual phase-shifted encoders for detection of speed and direction).

Using encapsulated motor systems has the advantage that the solution is much smaller than that using separate modules, plus the system is dust-proof and shielded against stray light, which is required for optical encoders. The disadvantage of using a fixed assembly like this is that the gear ratio may only be changed with difficulty, or not at all. A magnetic encoder comprises a disk equipped with several magnets and a Hall effect sensor [4].

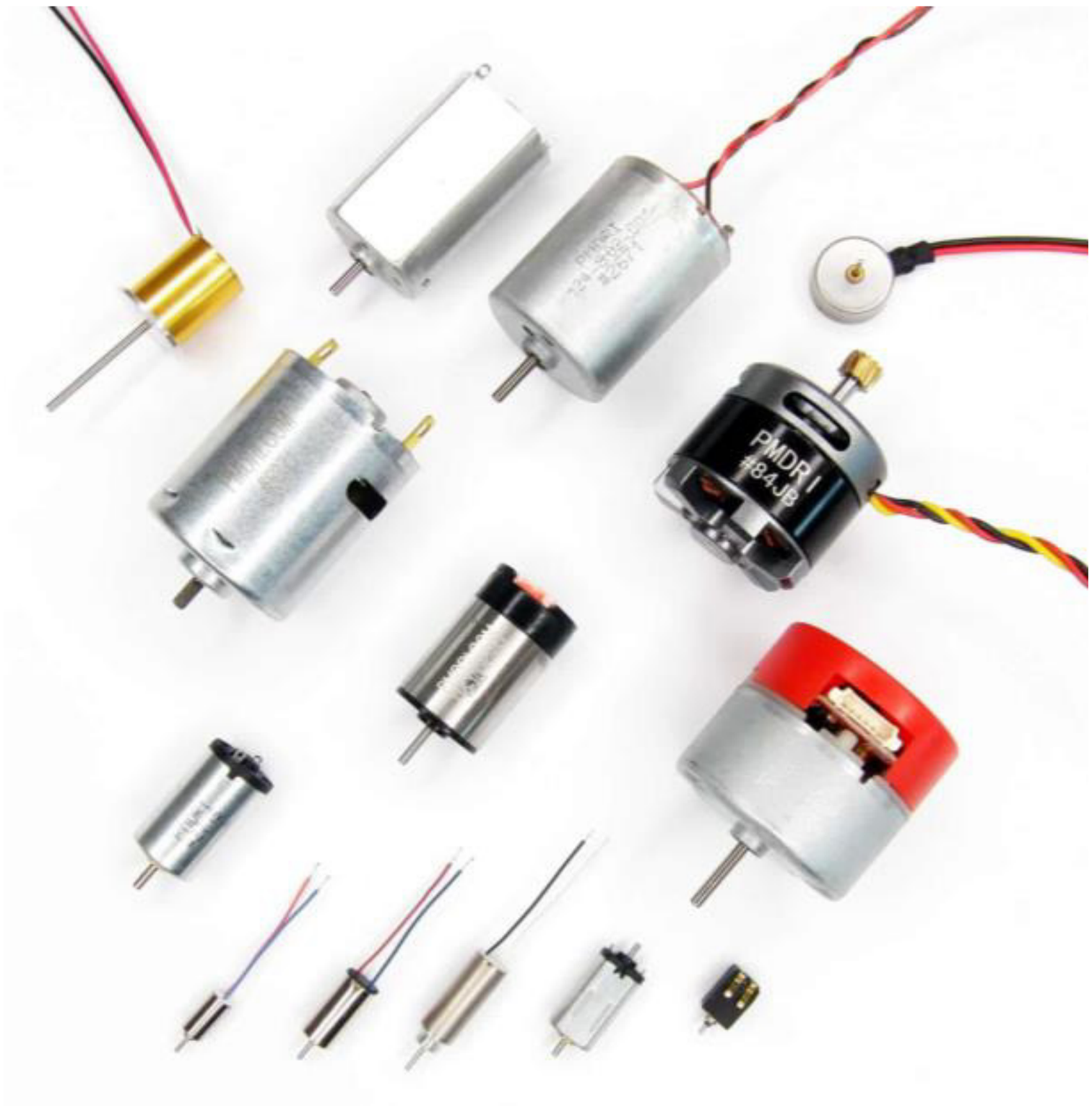


Figure 2.4 DC Motors Types

H Bridge is a simple electronic circuit that enables us to apply voltage to load in either direction. It is commonly used in robotics applications to control DC Motors. By using H Bridge, we can run DC Motor in clockwise or anticlockwise directions. Most mini-DC motors have two terminals because they are series type (armature and field windings are connected in series) DC motors [4].

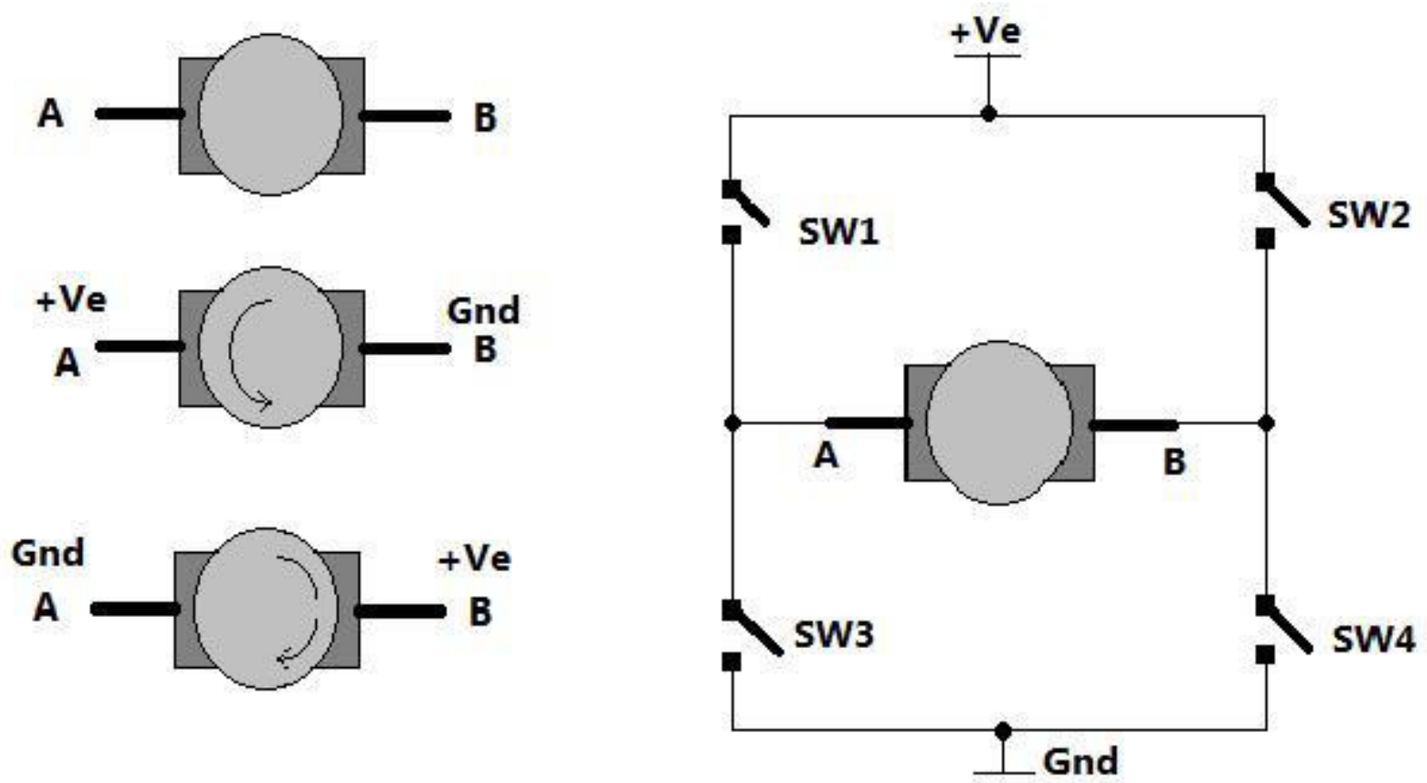


Figure 2.5 DC Motors H-Bridge.

If we connect terminal A with +Ve supply and terminal B with –Ve supply or ground the current will flow from the motor from A to B and the motor will rotate in one direction – say clockwise (CW) or forward direction. by changing the supply terminals. Now B is connected with +Ve and A is connected to the ground. The current will flow from the motor from B to A and the motor will rotate in another direction (counterclockwise – CCW or reverse).

Four switches are connected in between +Ve supply and ground and the DC motor is connected in between two switches as shown. Such circuit arrangement is known as H-bridge because it looks like the letter 'H' . If SW1 and SW4 are pressed simultaneously
then current will flow from +Ve – SW1 – A – B – SW4 – Gnd. So, the motor will rotate in one direction. Open (release) SW1 and SW4 to stop the motor.

Now if SW2 and SW3 are pressed current will flow from +Ve – SW2 – B – A – SW3 – Gnd. So, the motor gets reverse supply, and it will rotate in another direction. The circuit replaces the switches with NPN-type transistors. We all know that a transistor works as a switch. For an NPN transistor if we give +Ve input to the base it will turn on and if we give 0 input it will be turned OFF. So, in this circuit, if Q1 and Q4 are turned on simultaneously the motor will rotate forward and if Q2 and Q3 are turned on then the motor will rotate reverse. A motor can be rotated forward and reverse using 4 transistors only [4].

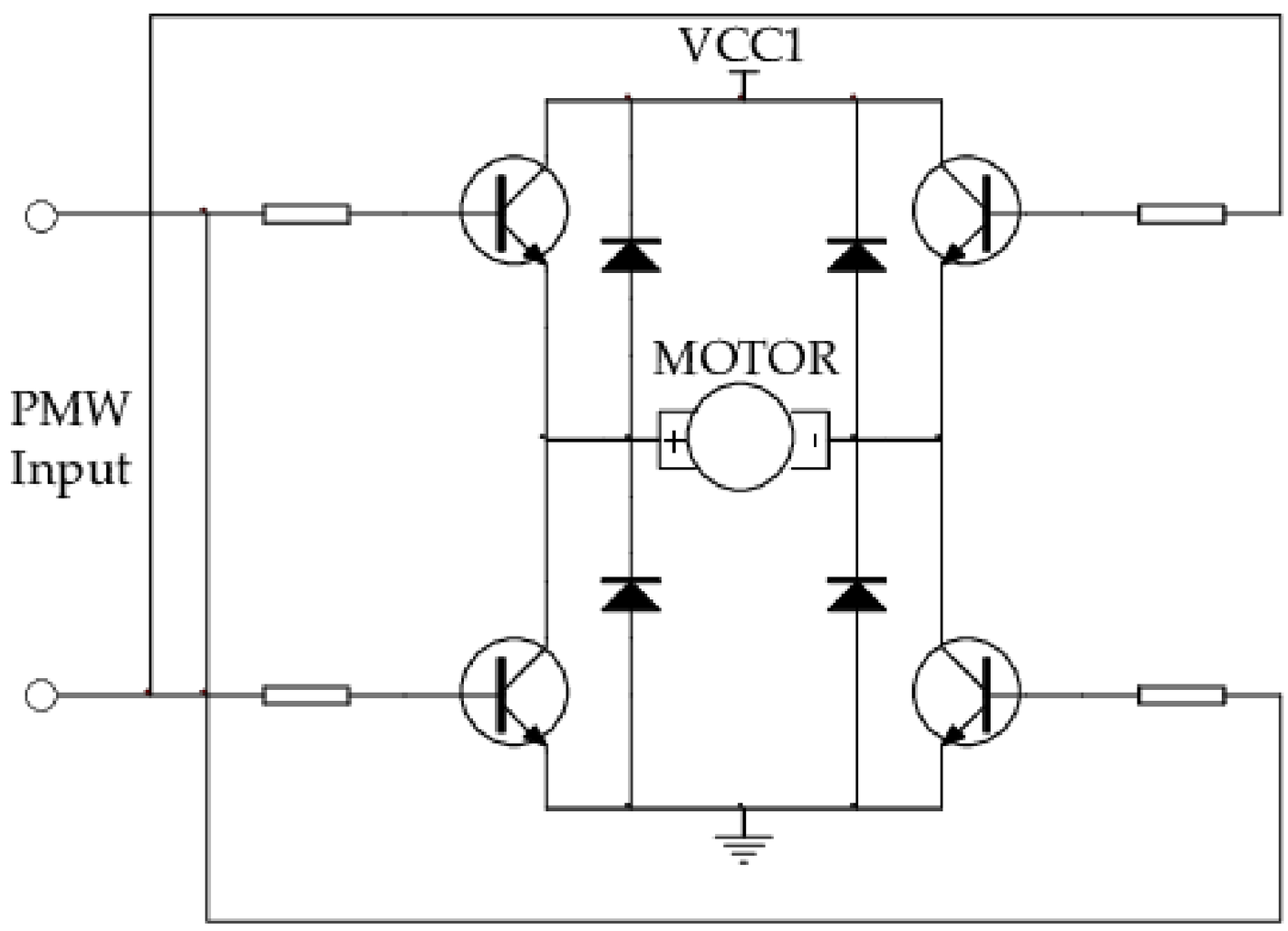


Figure 2.6 H-Bridge Circuit.

The DC motor speed varies as the applied input voltage varies. As you increase the applied input voltage the speed will increase and vice versa. Applying max rated voltage will rotate the motor at full speed. One of the very popular methods for generating variable DC voltage is pulse width modulation (PWM). Pulse width modulation means varying the width (duty) of the pulse. Width means ON time Ton of pulse. The average output voltage (Vdc or Vavg) is given by the equation:

$$V_{dc} = \frac{T_{on}}{T_{on} + T_{off}} * V_s$$

It directly depends upon Ton. If duty is 50% the average output voltage Vdc is exactly half of Vs. If duty is increased to 75%, Vdc also increases to 3/4th of Vs and if duty is decreased to 25%, Vdc reduces to 1/4th of Vs. Thus, as pulse width varies the average output voltage varies. So, we must apply PWM to the DC motor to vary its speed [4].

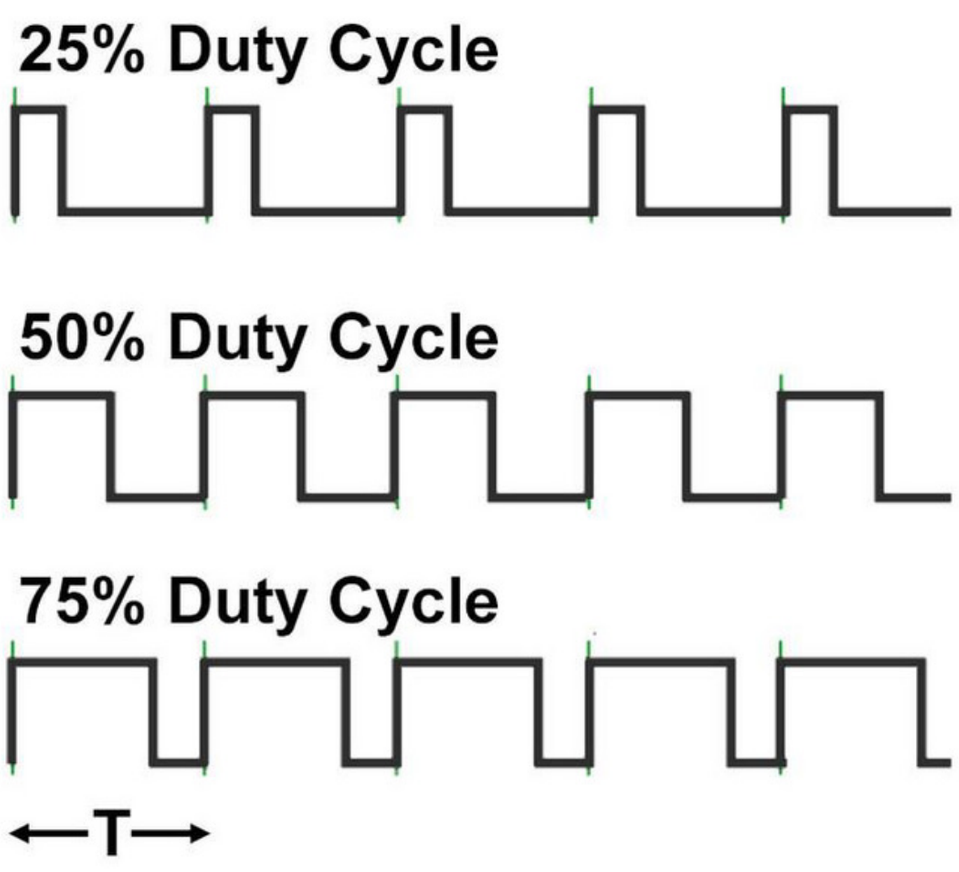


Figure 2.7 PWM Duty cycle.

2.1.1.c Power System:

As we say before, all hardware components need power to work, also not all hardware work at the same power level. Therefore, we must have a good power system to deliver all desired power to all hardware components. Therefore, we'll use Battery Management System. Battery management system (BMS) is technology dedicated to the oversight of a battery pack, which is an assembly of battery cells, electrically organized in a row x column matrix configuration to enable delivery of targeted range of voltage and current for a duration of time against expected load scenarios. The oversight that a BMS provides usually includes:

- Monitoring the battery.
- Providing battery protection.
- Estimating the battery's operational state.
- Continually optimizing battery performance.
- Reporting operational status to external devices.

The monitoring and control functions are specifically applied to individual cells, or groups of cells called modules in the overall battery pack assembly. Lithium-ion rechargeable cells have the highest energy density and are the standard choice for battery packs for many consumer products, from laptops to electric vehicles. While they perform superbly, they can be rather unforgiving if operated outside a generally tight safe operating area (SOA), with outcomes ranging from compromising the battery performance to outright dangerous consequences. The BMS certainly has a challenging job description, and its overall complexity and oversight outreach may span many disciplines such as electrical, digital, control, thermal, and hydraulic.

Here, we use small BMS to power the DC motors, sensors, and controllers. We'll be making a 12V 9000mAh Li-ion Battery pack in this post. We'll start by designing a 3s battery pack, then connecting the BMS to it to execute all the BMS's functions. Liion cells are increasingly used as battery packs for many applications due to their high energy density and rechargeable characteristics. However, we must link a Liion cell with a BMS to safeguard the circuit from being destroyed or reducing the cell's life. we'll construct a simple 3s battery pack and connect it to a 3s 45Amps BMS circuit.

Figure 2.8 Battery Parallel Connector.

The 18650 battery is a lithium-ion battery with a diameter of 18mm and a height of 65mm. Its height and diameter are both greater than the AA size. They are not compatible with AA or AAA size batteries. Because of its high-level capabilities, such as 250+ charge cycles and increased energy density, the 18650-battery type is useful in rechargeable and high-current drain devices. Material Required for a 12V Li-ion Battery Pack:

- 18650 Li-ion Cells x9
- 3S 45Amp BMS (Battery Management System)
- 0.15mm Coated Nickle Strips
- Parallel Connector
- JST XH 2.54 Female 4-Pin Connector

Every 18650 cell can be charged up to 4.2V; we need three cells in series to make a 12.6V battery pack. In the figure above, the connections are indicated.

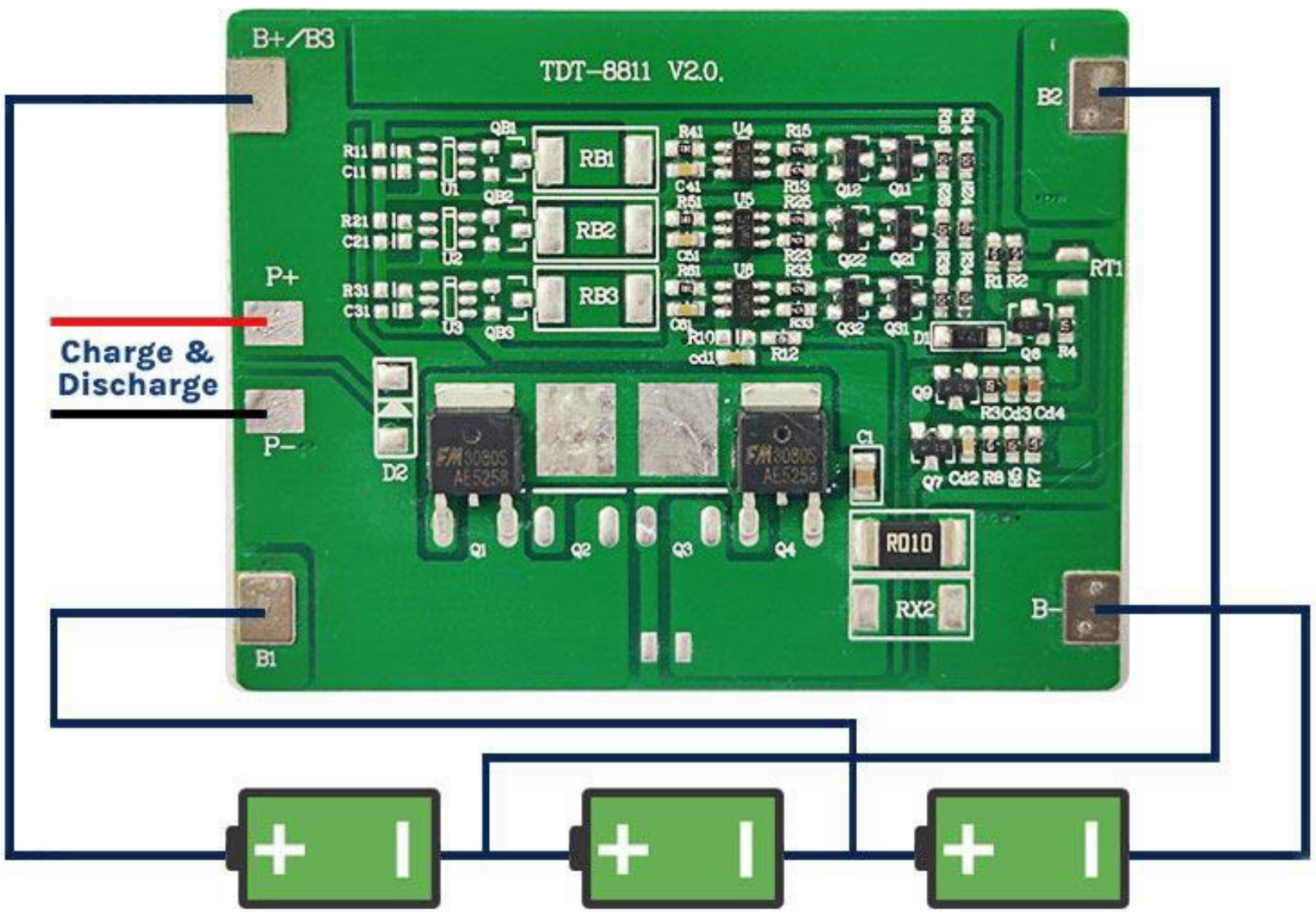


Figure 2.9 Connections for 12V Battery Pack with BMS.

Marking On the BMS	Connection with the BMS
P+	Connection to the battery pack's positive terminal for charging and attaching the load
P-	Connection to the battery pack's negative terminal for charging and attaching the load
B-	Negative terminal of the 1st cell
B1	Positive terminal of the 1st cell
B2	Positive terminal of the 2nd cell
B+/B3	Positive terminal of the 3rd cell

table 2.2 Connection with the BMS

2.2 Software Design

Now as we mension: we will taking software design description the software architecture layers and interfacing functions used at each block mensioed at each layer.

The layers used at architecture design are:

- APP layer (Application Layer)
- HAL (Hardware Abstraction Layer)
- MCAL (Microcontroller Abstraction Layer)
- LIB layer (Library Layer)

2.2.1 Abstraction Layers Design:

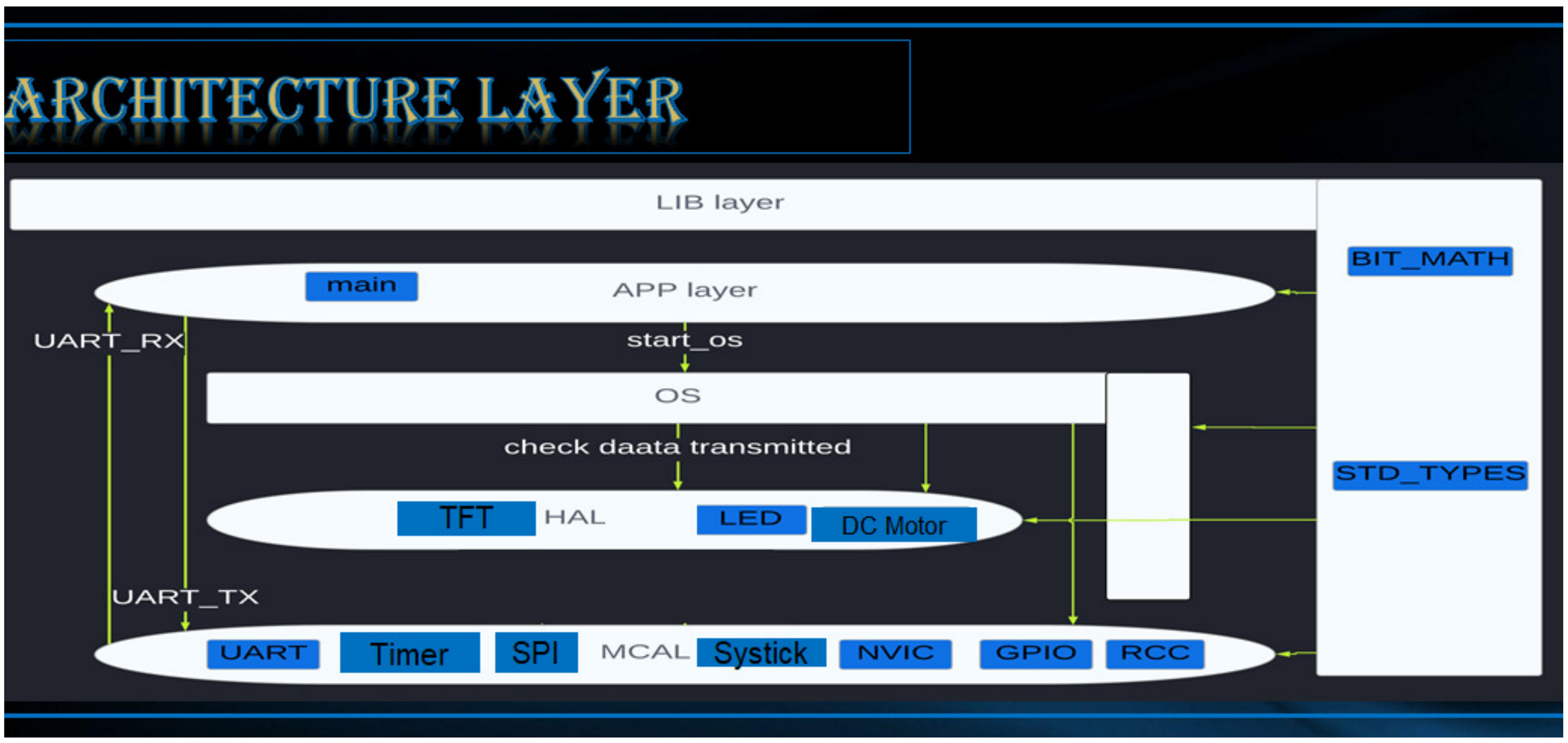


Figure 2.10 Architecture layer of acceleration subsystem

2.2.2 Internal SW Architecture APIS:

- GPIO APIs description
- RCC APIs description
- NVIC APIs description
- SPI APIs description
- UART APIs description
- SYSTIC APIs description
- Timer APIs description

2.2.2.a GPIO APIs description:

Function Description : select mode of pin (output or input or analog or AF)

```
void MGPIO_voidSetPinMode (u8 A_u8PortId , u8 A_u8PinNumber , u8 A_u8Mode) ;
```

Function Description : select the pin is push pull or open drain

```
void MGPIO_voidSetPinOutputType (u8 A_u8PortId       ,
                                 u8 A_u8PinNumber    ,
                                 u8 A_u8OutputType) ;
```

Function Description : select speed of mc

```
void MGPIO_voidSetPinOutputSpeed (u8 A_u8PortId       ,
                                  u8 A_u8PinNumber    ,
                                  u8 A_u8OutputSpeed) ;
```

Function Description : select input pullup or pulldown or non pullup

```
void MGPIO_voidSetPinInputPull (u8 A_u8PortId       ,
                                u8 A_u8PinNumber ,
                                u8 A_u8InputPull)   ;
```

Function Description : set pin to low or high

```
void MGPIO_voidSetPinValue (u8 A_u8PortId , u8 A_u8PinNumber ,
                            u8  A_u8PinValue) ;
```

Function Description : tog pin statues

```
void MGPIO_voidTogPinValue (u8 A_u8PortId , u8 A_u8PinNumber) ;
```

Function Description : read the statues of pin

```
u8 MGPIO_voidGetPinValue (u8 A_u8PortId , u8 A_u8PinNumber) ;
```

Function Description : initialize the GPIO pin as alternative function

```
void MGPIO_voidSetAltFun (u8 A_u8PortId , u8 A_u8PinNumber , u8 A_u8AltFun) ;
```

Function Description : initialize the GPIO Peripheral

```
void MGPIO_voidInit (MGPIO_Config_t * A_PstructPinConfig ,
                     u8                A_u8Port          ,
                     u8                A_u8Pin)          ;
```

2.2.2.b RCC APIs description:

Function Description : initialize the RCC Peripheral
void MRCC_voidInit (void) ;

Function Description : enable the selected Peripheral
void MRCC_voidEnablePerephiralClock (u32 A_u32BusId, u32 A_u32PeripheralId) ;

Function Description : disable the selected Peripheral
void MRCC_voidDisablePerephiralClock (u32 A_u32BusId, u32 A_u32PeripheralId) ;

2.2.2.c NVIC APIs description:

Function Description : enable PIE of peripheral
void MNVIC_voidEnablePeripheral (u8 A_u8InterruptId) ;

Function Description : disable PIE of peripheral
void MNVIC_voidDisablePeripheral (u8 A_u8InterruptId) ;

Function Description : enable pending flag
void MNVIC_voidSetPendingFlag (u8 A_u8InterruptId) ;

Function Description : disable pending flag
void MNVIC_voidClrPendingFlag (u8 A_u8InterruptId) ;

Function Description : read statues of active flag
u8 MVNIC_u8GetActiveFlagStatues (u8 A_u8InterruptId) ;

Function Description : set the groups and sub priority config
Options : GROUP_4BITS , GROUP_3BITS , GROUP_2BITS , GROUP_1BITS ,
GROUP_0BITS
void MNVIC_voidSetPriorityConfig(u8 A_u8PriorityOption) ;

2.2.2.d SPI APIs description:

Function Description : initialize the selected SPI
void MSPI_voidInit (u8 A_u8SpiType) ;

Function Description : send and transceive data
u32 MSPI_u32Transceive(u8 A_u8SpiType , u32 A_u32Data) ;

2.2.2.e UART APIs description:

Function Description : initialize registers of UART

```
void MUART_voidInit (MUART_MemMap_t *A_xUART_Type,
                     MUART_Init_t *        A_xUART_ClockInitStruct ,
                     MUART_ClockInit_t * A_xUART_InitStruct) ;
```

Function Description : Disable UART

```
void MUART_voidDisable (MUART_MemMap_t *A_xUART_Type) ;
```

Function Description : Transmit one byte Synchronous

```
void MUART_voidTransmitByte (MUART_MemMap_t * A_xUART_Type ,
                             u8                          A_u8Byte) ;
```

Function Description : Transmit string Synchronous

```
void MUART_voidTransmitString (MUART_MemMap_t *A_xUART_Type ,
                               u8                          *A_u8String) ;
```

Function Description : receive one byte Synchronous non blocking

```
u8 MUART_u8ReciveByteSyncNonBlocking (MUART_MemMap*A_xUART_Type) ;
```

2.2.2.f Timers APIs description:

Function Description : initialize selected channel Timer

```
void MTimer_voidInt (void) ;
```

Function Description : set pwm for selected channel Timer

```
void MTimer_voidPWM (u8 A_u8TimerSelected       ,
                     u8 A_u8Channel             ,
                     u16 A_u16CompareValue) ;
```

2.2.2.g Systic APIs description:

Function Description : initial the clock source of systick

```
void MSTK_voidInit (void) ;
```

Function Description : stop timer

```
void MSTK_voidStopTimer (void) ;
```

Function Description : delay with selected time

```
void MSTK_voidSetBusyWait (f32 A_f32TimeInMill) ;
```

Function Description : go to handler single time

```
void MSTK_voidSetIntervalSingle (f32 A_f32TimeInMill , void (*fptr)(void)) ;
```

2.2.3 External SW Architecture APIS:

- LED APIs description
- TFT APIs description
- PWM_DC motors APIs description

2.2.3.a LED APIs description:

Function Description : Define the led as OUTPUT

```
void HLED_voidLedInitial (u8 A_u8LED_ID) ;
```

Function Description : Turn on LED

```
void HLED_voidLedOn (u8 A_u8LED_ID) ;
```

Function Description : Turn off LED

```
void HLED_voidLedOff (u8 A_u8LED_ID) ;
```

Function Description : Reverse the status of LED

```
void HLED_voidLedToggle (u8 A_u8LED_ID) ;
```

2.2.3.b TFT APIs description:

Function Description : initialize tft

```
void HTFT_voidInit (u8 A_u8SpiType) ;
```

Function Description : display image

```
void HTFT_voidDisplayImage (u8 A_u8SpiType , const u16 AR_u16ImageArray []  ,
                            u32  A_u32ArraySize) ;
```

Function Description : drow string started from row and col selected

```
void HTFT_voidDisplayString (u8 A_u8SpiType , s8  A_s8L_u8Row,  s8  A_s8Col ,
                             u8  A_s8String [] , u16 A_u16Foreground ) ;
```

2.2.3.b TFT APIs description:

Function Description : initialize DC

```
void HDC_voidInt  (void) ;
```

Function Description : rotate DC motor in counter wise

```
void HDC_voidRoCw (void) ;
```

Function Description : rotate DC motor in counter clock wise

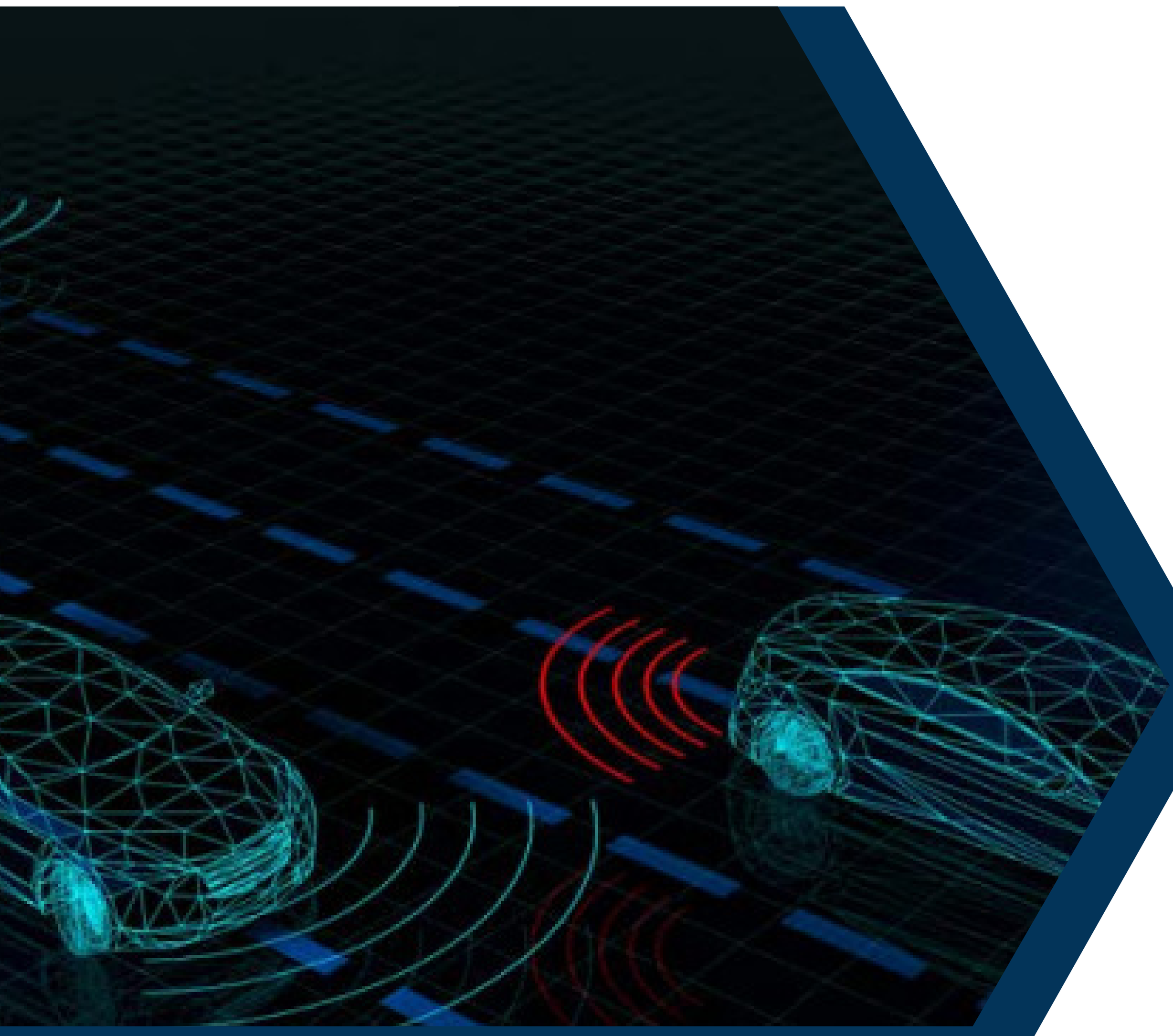
```
void HDC_voidRoCcw (void) ;
```

Function Description : turn off DC motor

```
void HDC_voidStop (void) ;
```

B

LINUX SYSTEM



Chapter 3: Communication Subsystem
Chapter 4: Embedded Linux Subsystem
Chapter 5: OpenCV Subsystem

Communication Subsystem

3

Serial communication is a method of data transfer between two electronic devices by sending and receiving data one bit at a time over a single communication line. In contrast to parallel communication, where multiple data bits are transmitted simultaneously over multiple communication lines, serial communication uses a single line to transmit data.

Serial communication can be classified into two categories: synchronous and asynchronous communication.

Synchronous communication uses a clock signal shared between the two devices to synchronize the transmission of data, whereas asynchronous communication uses a start bit, stop bit, and a fixed number of data bits to indicate the beginning and end of a data transmission [5].

Serial communication is commonly used in many electronic devices, such as microcontrollers, computers, and peripherals, to exchange data over short distances. For example, a microcontroller can use serial communication to communicate with other microcontrollers, sensors, or actuators in a system [5].

One of the advantages of serial communication is its simplicity and low cost. Because it uses a single communication line, serial communication requires fewer wires and simpler hardware than parallel communication. Additionally, serial communication can be used over longer distances than parallel communication, making it a useful option in situations where data needs to be transmitted over greater distances.

In summary, serial communication is a method of data transfer between two electronic devices that sends and receives data one bit at a time over a single communication line. It is commonly used in many electronic devices due to its simplicity, low cost, and the ability to transmit data over longer distances than parallel communication.

Used in my project:

- **Wire communication:** UART, SPI
- **Wireless communication:** Bluetooth, WIFI, IR

3.1 Wire communication:

Wired communication refers to the transmission of data over a wire-based communication technology. Wired communication is also known as wireline communication.

3.1.1 USART:

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator. It supports synchronous one-way communication and half-duplex single wire communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication. High speed data communication is possible by using the DMA for multibuffered configuration [6].

3.1.1.a UART main features:

- Full duplex, asynchronous communications
- Configurable oversampling method by 16 or by 8 to give flexibility between speed and clock tolerance
- Fractional baud rate generator systems
- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- Separate enable bits for transmitter and receiver
- Parity control:
  – Transmits parity bit – Checks parity of received data byte
- Four error detection flags:
  – Overrun error – Noise detection – Frame error – Parity error

3.1.1.b UART Specs:

1. Wired
2. Serial
3. Peer to peer
4. Asynchronous
5. Half or full duplex

3.1.1.c UART Hardware interface:

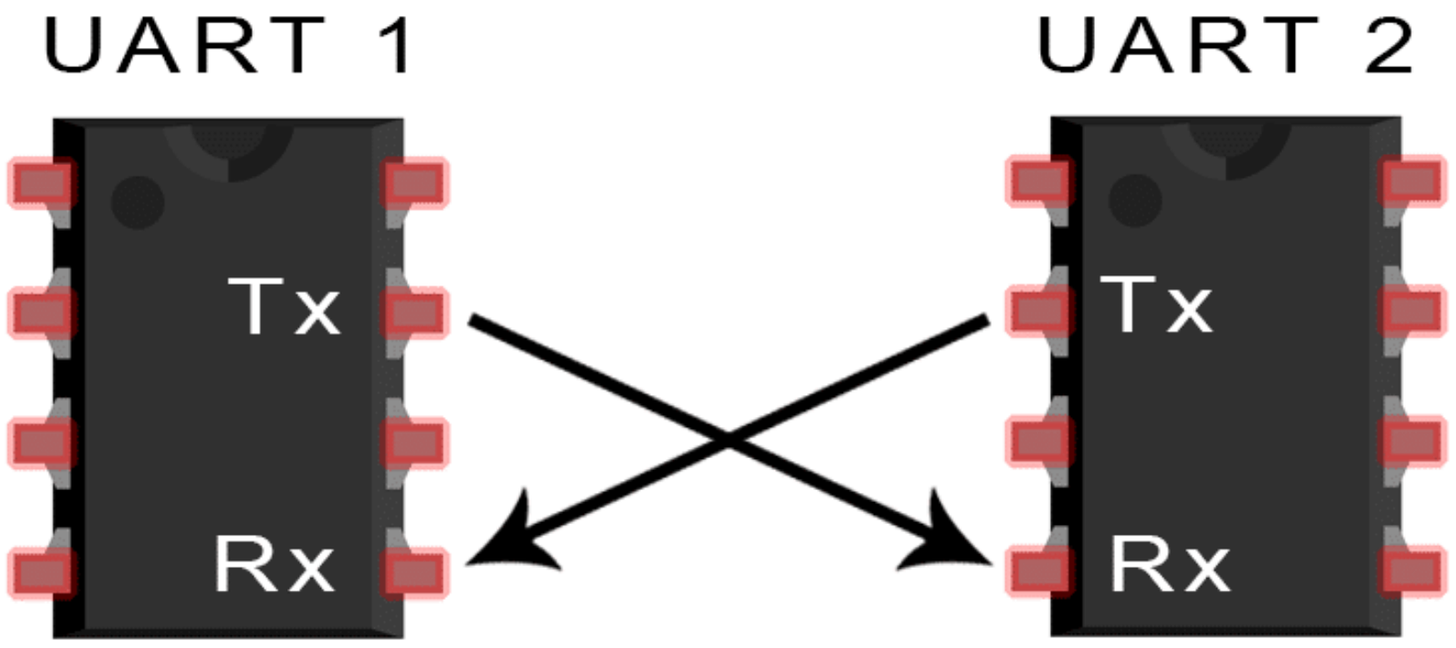


Figure 3.1 UART Hardware interface

3.1.1.d USART functional description:

The interface is externally connected to another device by three pins (see Figure 167). Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW_RX).

Through these pins, serial data is transmitted and received in normal USART mode as frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)
- Data Register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guard time Register (USART_GTPR) in case of Smartcard mode [6].

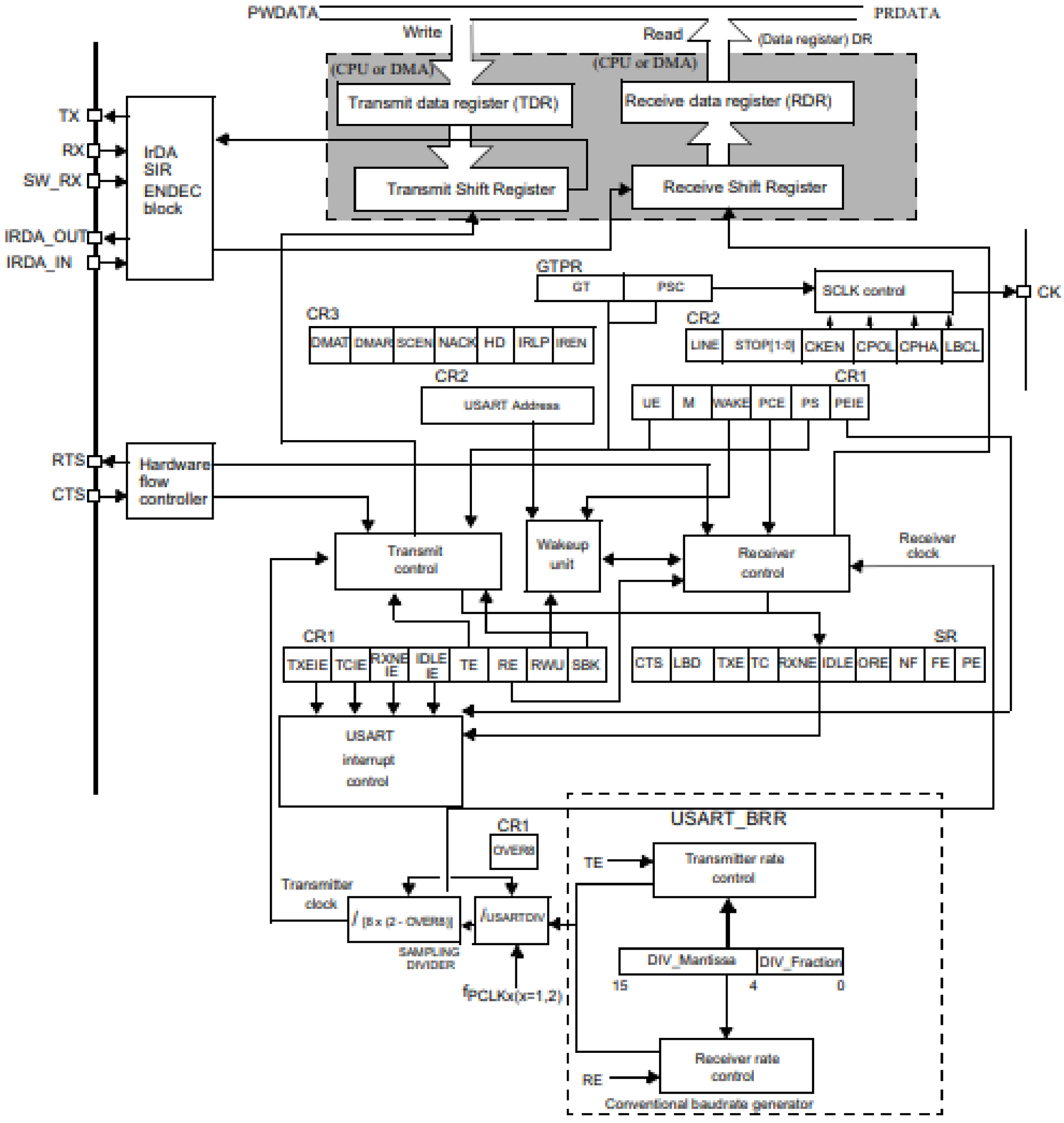


Figure 3.2 UART Block diagram

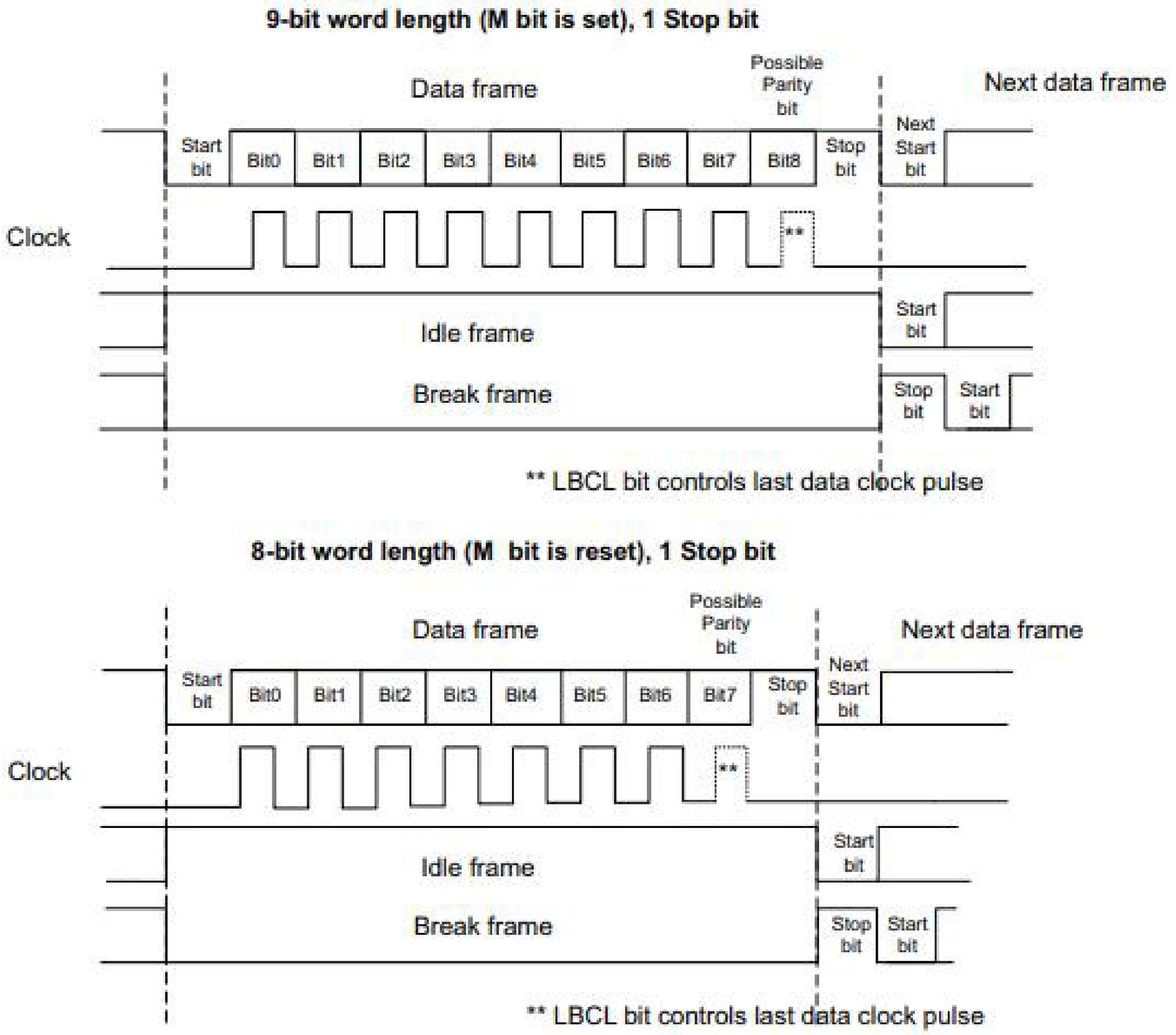


Figure 3.3 Word length programming

3.1.1.e USART Transmitter:

The transmitter can send data words of either 8 or 9 bits depending on the M bit status. When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

Character transmission:

During an USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register, every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits [6].

The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.

Note: The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen.
The current data being transmitted will be lost.
An idle frame will be sent after the TE bit is enabled.
Configurable stop bits
the number of stop bits to be transmitted with every character can be programmed in Control register 2, bits 13,12.

- 1 stop bit: This is the default value of number of stop bits.
- 2 Stop bits: This will be supported by normal USART, single-wire and modem modes.
- 0.5 stop bit: To be used when receiving data in Smartcard mode.
- 1.5 stop bits: To be used when transmitting and receiving data in Smartcard mode.

An idle frame transmission will include the stop bits.
A break transmission will be 10 low bits followed by the configured number of stop bits (when m = 0) and 11 low bits followed by the configured number of stop bits (when m = 1). It is not possible to transmit long breaks (break of length greater than 10/11 low bits)

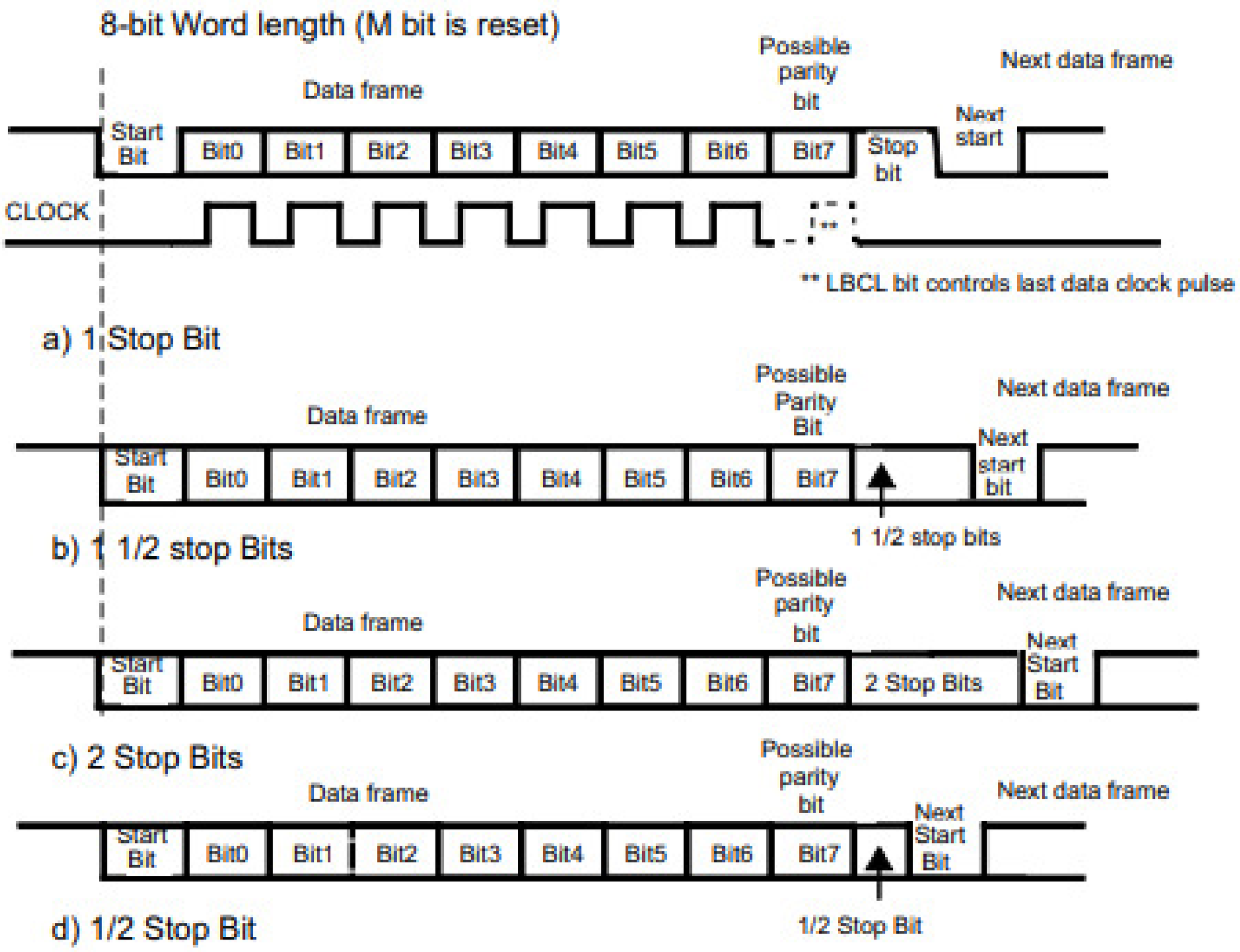


Figure 3.4 Configurable stop bits

Procedure:

- Enable the USART by writing the UE bit in USART_CR1 register to
- Program the M bit in USART_CR1 to define the word length.
- Program the number of stop bits in USART_CR2.
- Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
- Select the desired baud rate using the USART_BRR register.
- Set the TE bit in USART_CR1 to send an idle frame as first transmission.
- Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
- After writing the last data into the USART_DR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission [6].

Single byte communication

Clearing the TXE bit is always performed by a write to the data register.

The TXE bit is set by hardware and it indicates:

- The data has been moved from TDR to the shift register and the data transmission has started.
- The TDR register is empty.
- The next data can be written in the USART_DR register without overwriting the previous data.

This flag generates an interrupt if the TXEIE bit is set.

When a transmission is taking place, a write instruction to the USART_DR register stores the data in the TDR register and which is copied in the shift register at the end of the current transmission.

When no transmission is taking place, a write instruction to the USART_DR register places the data directly in the shift register, the data transmission starts, and the TXE bit is immediately set.

If a frame is transmitted (after the stop bit) and the TXE bit is set, the TC bit goes high. An interrupt is generated if the TCIE bit is set in the USART_CR1 register.

After writing the last data into the USART_DR register, it is mandatory to wait for TC=1 before disabling the USART or causing the microcontroller to enter the low-power mode (see Figure 170: TC/TXE behavior when transmitting) [6].

The TC bit is cleared by the following software sequence:

- A read from the USART_SR register
- A write to the USART_DR register

Note: The TC bit can also be cleared by writing a ‘0 to it. This clearing sequence is recommended only for Multibuffer communication.

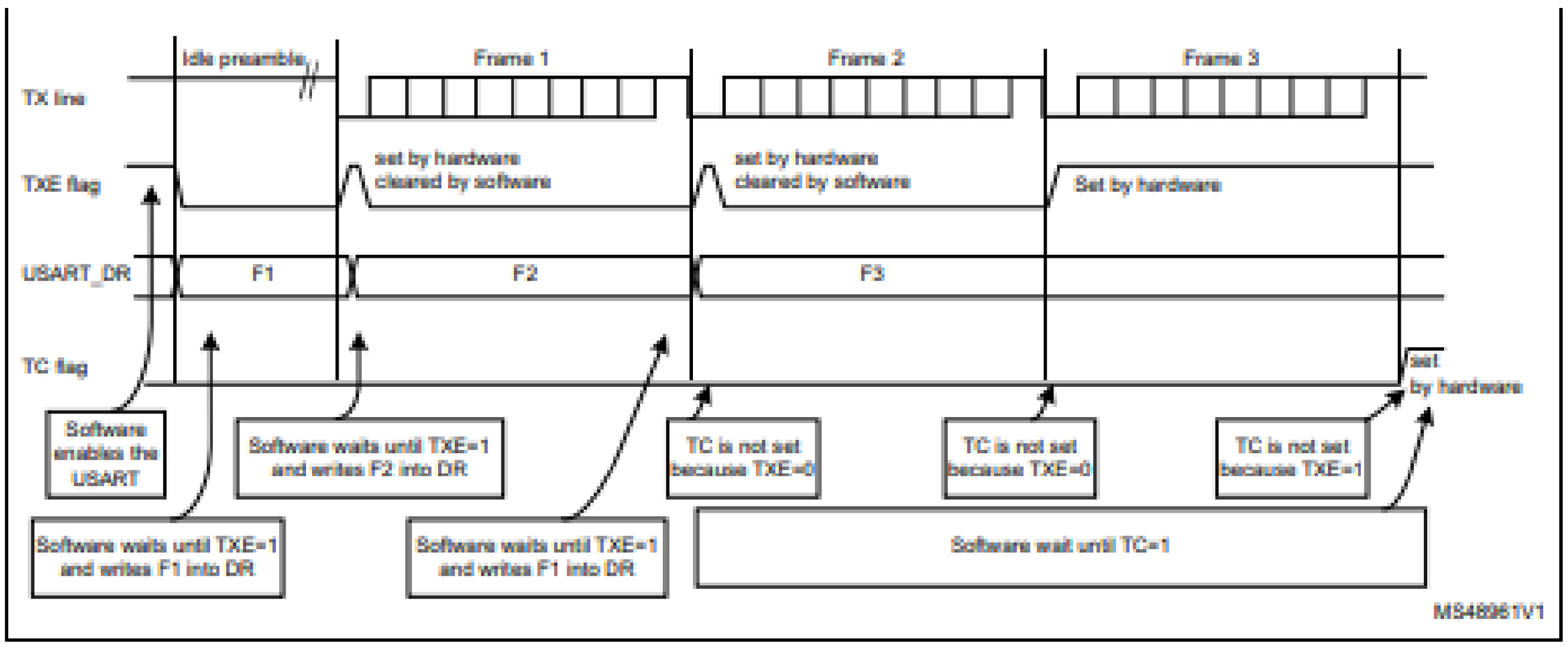


Figure 3.5 TC/TXE behavior when transmitting

Break characters

Setting the SBK bit transmits a break character. The break frame length depends on the M bit (see Figure 168).

If the SBK bit is set to ‘1 a break character is sent on the TX line after completing the current character transmission. This bit is reset by hardware when the break character is completed (during the stop bit of the break character). The USART inserts a logic 1 bit at the end of the last break frame to guarantee the recognition of the start bit of the next frame [6].

Note: If the software resets the SBK bit before the commencement of break transmission, the break character will not be transmitted. For two consecutive breaks, the SBK bit should be set after the stop bit of the previous break.

Idle characters

Setting the TE bit drives the USART to send an idle frame before the first data frame.

3.1.1.f USART Receiver:

The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Start bit detection

The start bit detection sequence is the same when oversampling by 16 or by 8.

In the USART, the start bit is detected when a specific sequence of samples is recognized. This sequence is: 1 1 1 0 X 0 X 0 X 0 0 0 0

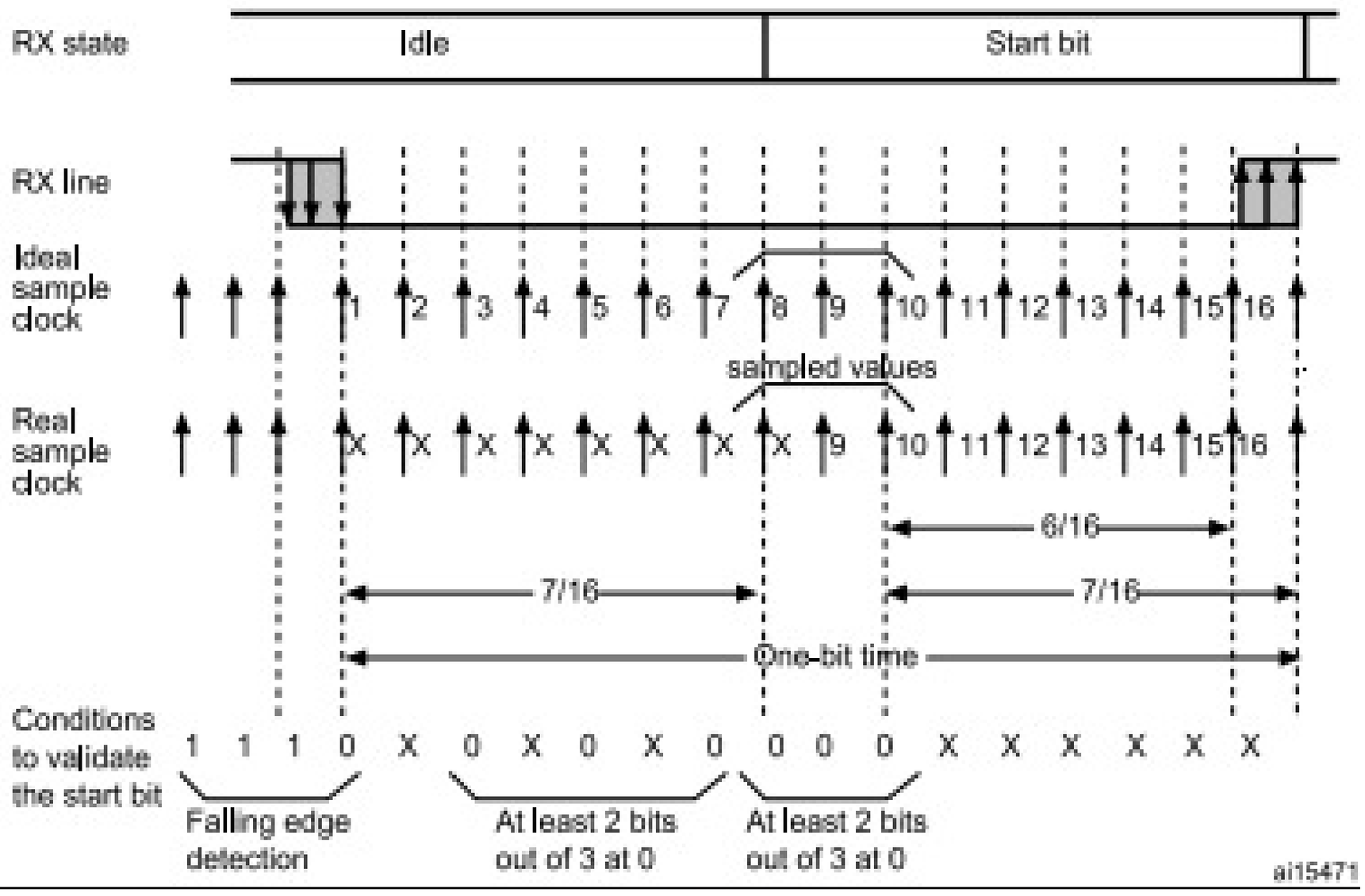

RX state
Idle
Start bit
RX line
Ideal sample clock
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
sampled values
Real sample clock
x x x x x x x x x 9 10 11 12 13 14 15 16
6/16
7/16
7/16
One-bit time
Conditions to validate the start bit
1 1 1 0 X 0 X 0 X 0 0 0 0 X X X X X X
Falling edge detection
At least 2 bits out of 3 at 0
At least 2 bits out of 3 at 0
ai15471

Figure 3.6 start bit detection when oversampling by 16 or 8

Note: If the sequence is not complete, the start bit detection aborts and the receiver return to the idle state (no flag is set) where it waits for a falling edge.

The start bit is confirmed (RXNE flag set, interrupt generated if RXNEIE=1) if the 3 sampled bits are at 0 (first sampling on the 3rd, 5th and 7th bits finds the 3 bits at 0 and second sampling on the 8th, 9th and 10th bits also finds the 3 bits at 0).

The start bit is validated (RXNE flag set, interrupt generated if RXNEIE=1) but the NE noise flag is set if, for both samplings, at least 2 out of the 3 sampled bits are at 0 (sampling on the 3rd, 5th and 7th bits and sampling on the 8th, 9th and 10th bits).

If this condition is not met, the start detection aborts and the receiver returns to the idle state (no flag is set). If, for one of the samplings (sampling on the 3rd, 5th and 7th bits or sampling on the 8th, 9th and 10th bits), 2 out of the 3 bits are found at 0, the start bit is validated but the NE noise flag bit is set [6].

Character reception

During an USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register. Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication. STEP 3
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received:

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error [6].

Note: The RE bit should not be reset while receiving data. If the RE bit is disabled during reception, the reception of the current byte will be aborted.

Break character
When a break character is received, the USART handles it as a framing error.
Idle character.

When an idle frame is detected, there is the same procedure as a data received character plus an interrupt if the IDLEIE bit is set.

3.1.1.g USART Overrun error:

An overrun error occurs when a character is received when RXNE has not been reset. Data cannot be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced.
When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_DR is performed.
- The shift register will be overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or both the EIE and DMAR bits are set.
- The ORE bit is reset by a read to the USART_SR register followed by a USART_DR register read operation. Note: The ORE bit, when set, indicates that at least 1 data has been lost. There are two possibilities:
- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,
- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received. It may also occur when the new data is received during the reading sequence (between the USART_SR register read access and the USART_DR read access) [6].

3.1.1.h USART Fractional baud rate generation:

An overrun error occurs when a character is received when RXNE has not been reset. Data cannot be transferred from the shift register to the RDR register until the RXNE bit is cleared.

The RXNE flag is set after every byte received. An overrun error occurs if RXNE flag is set when the next data is received or the previous DMA request has not been serviced.
When an overrun error occurs:

- The ORE bit is set.
- The RDR content will not be lost. The previous data is available when a read to USART_DR is performed.
- The shift register will be overwritten. After that point, any data received during overrun is lost.
- An interrupt is generated if either the RXNEIE bit is set or both the EIE and DMAR bits are set.
- The ORE bit is reset by a read to the USART_SR register followed by a USART_DR register read operation. Note: The ORE bit, when set, indicates that at least 1 data has been lost. There are two possibilities:
- if RXNE=1, then the last valid data is stored in the receive register RDR and can be read,
- if RXNE=0, then it means that the last valid data has already been read and thus there is nothing to be read in the RDR. This case can occur when the last valid data is read in the RDR at the same time as the new (and lost) data is received. It may also occur when the new data is received during the reading sequence (between the USART_SR register read access and the USART_DR read access) [6].

3.1.1.i USART Parity control:

- Parity control (generation of parity bit in transmission and parity checking in reception) can be enabled by setting the PCE bit in the USART_CR1 register. Depending on the frame length defined by the M bit, the possible USART frame formats are as listed in Table.

M bit	PCE bit	USART frame(1)
0	0	| SB | 8 bit data | STB |
0	1	| SB | 7-bit data | PB | STB |
1	0	| SB | 9-bit data | STB |
1	1	| SB | 8-bit data PB | STB |

Table 3.1 Frame format

Even parity

The parity bit is calculated to obtain an even number of "1s" inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit. E.g.: data=00110101; 4 bits set => parity bit will be 0 if even parity is selected (PS bit in USART_CR1 = 0).

Odd parity

The parity bit is calculated to obtain an odd number of "1s" inside the frame made of the 7 or 8 LSB bits (depending on whether M is equal to 0 or 1) and the parity bit. E.g.: data=00110101; 4 bits set => parity bit will be 1 if odd parity is selected (PS bit in USART_CR1 = 1).

Parity checking in reception

If the parity check fails, the PE flag is set in the USART_SR register and an interrupt is generated if PEIE is set in the USART_CR1 register. The PE flag is cleared by a software sequence (a read from the status register followed by a read or write access to the USART_DR data register).

Note: In case of wakeup by an address mark: the MSB bit of the data is taken into account to identify an address but not the parity bit. And the receiver does not check the parity of the address data (PE is not set in case of a parity error).

Parity generation in transmission

If the PCE bit is set in USART_CR1, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit (even number of "1s" if even parity is selected (PS=0) or an odd number of "1s" if odd parity is selected (PS=1)).

Note: The software routine that manages the transmission can activate the software sequence which clears the PE flag (a read from the status register followed by a read or write access to the data register). When operating in half-duplex m

3.1.1.j USART Interrupts:

Interrupt event	Event flag	Enable control bit
Transmit Data Register Empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multibuffer communication	NF or ORE or FE	EIE

Table 3.2 USART interrupt requests

The USART interrupt events are connected to the same interrupt vector (see Figure 191).

- During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.
- While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

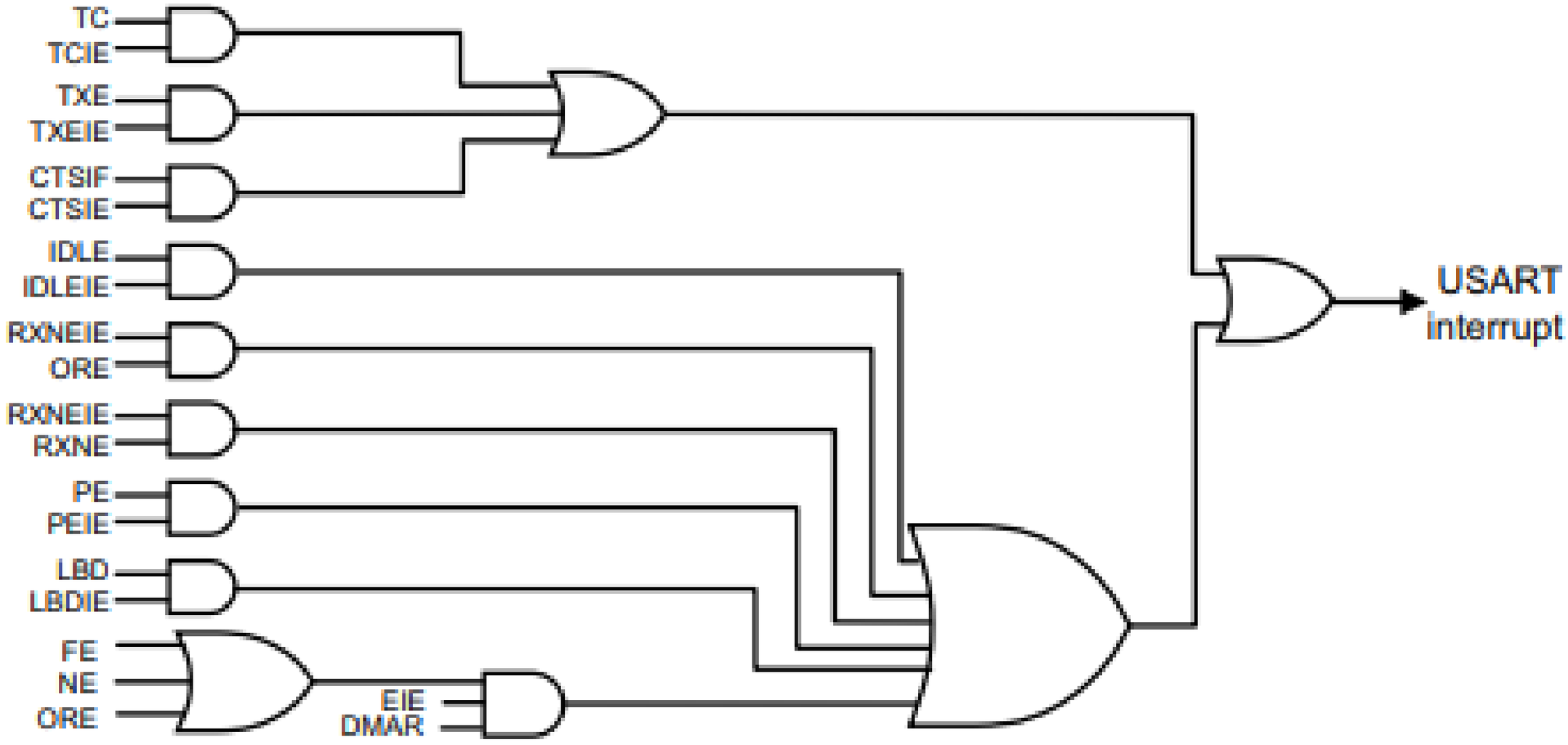


Figure 3.7 USART interrupt mapping diagram

3.1.2 SPI:

The SPI interface provides two main functions, supporting either the SPI protocol or the I2S audio protocol. By default, it is the SPI function that is selected. It is possible to switch the interface from SPI to I2S by software.

The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multimaster configuration.

It may be used for a variety of purposes, including simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication using CRC checking [6].

The I2S is also a synchronous serial communication interface. It can address four different audio standards including the I2S Philips standard, the MSB- and LSB-justified standards, and the PCM standard. It can operate as a slave or a master device in full-duplex mode (using 4 pins) or in half-duplex mode (using 3 pins). Master clock can be provided by the interface to an external slave component when the I2S is configured as the communication master The SPI protocol uses four signal lines:

1. MOSI (Master Output Slave Input): The master sends data to the slave on this line. The slave reads the data on this line.
2. MISO (Master Input Slave Output): The slave sends data to the master on this line. The master reads the data on this line.
3. SCLK (Serial Clock): The master sends clock signals on this line to synchronize the data transfer between the master and the slave.
4. SS (Slave Select): This signal line is used by the master to select the slave device with which it wishes to communicate.
5. The SPI protocol has various configuration options, including the clock speed, clock phase and polarity, and the number of data bits per transfer. These options can be adjusted to optimize the data transfer rate and ensure reliable communication between the devices.

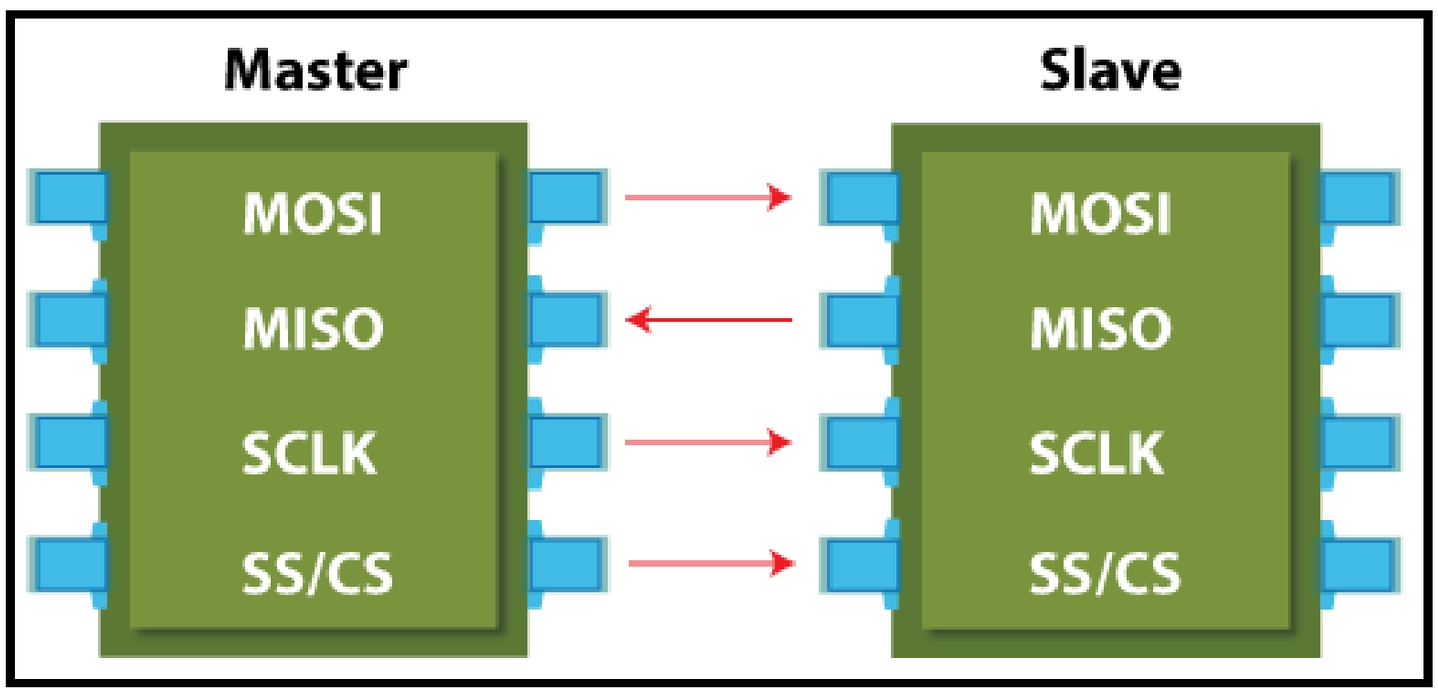


Figure 3.8 SPI pinout

3.1.2.a SPI Hardware:

To use the SPI protocol for communication between two devices, you need to connect the four SPI signal lines between the two devices. Here's how you can do it:

1. Connect the MOSI pin of the master device to the MOSI pin of the slave device.
2. Connect the MISO pin of the master device to the MISO pin of the slave device.
3. Connect the SCLK pin of the master device to the SCLK pin of the slave device.
4. Connect the SS pin of the master device to the SS pin of the slave device.

It's important to note that the SS (Slave Select) pin needs to be connected to a different pin on each slave device. This is because the master device uses the SS signal to select the specific slave device it wants to communicate with, and only thatslave device should respond to the communication [6].

Additionally, you'll need to make sure that the voltage levels of the SPI signals are compatible between the two devices. It's important to check the datasheets of both devices to ensure that the voltage levels match or can be easily converted using level shifters or voltage dividers.

Once the hardware connections are made, you can start using the SPI protocol to exchange data between the two devices. The specific details of how to use the SPI protocol will depend on the programming language and hardware platform that you're using.

3.1.2.b main features of the SPI protocol:

- Full-duplex synchronous transfers on three lines
- Simplex synchronous transfers on two lines with or without a bidirectional data line
- 8- or 16-bit transfer frame format selection
- Master or slave operation
- Multimaster mode capability
- 8 master mode baud rate prescalers (fPCLK/2 max.)
- Slave mode frequency (fPCLK/2 max)
- Faster communication for both master and slave
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- SPI TI mode
- Hardware CRC feature for reliable communication: – CRC value can be transmitted as last byte in Tx mode – Automatic CRC error checking for last received byte
- Master mode fault, overrun and CRC error flags with interrupt capability
- 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

3.1.2.c SPI Specs:

1. Data transfer rate: The maximum data transfer rate for the SPI protocol depends on the clock speed and the number of data bits per transfer. Typical clock speeds range from a few kHz up to several MHz, with data rates of up to tens or hundreds of megabits per second.
2. Clock polarity and phase: The SPI protocol allows for two different clock polarities (CPOL) and clock phases (CPHA), which determine when data is sampled and transmitted. The CPOL and CPHA settings can be configured to suit the specific requirements of the devices being used.
3. Data order: The SPI protocol can transmit data in either MSB (most significant bit) or LSB (least significant bit) first order. The data order can also be configured to suit the specific requirements of the devices being used.
4. Slave select: The SPI protocol supports multiple slave devices, and each slave is typically selected using a dedicated Slave Select (SS) signal line. The SS signal can be active low or active high, depending on the specific device being used.
5. Hardware overhead: The SPI protocol has relatively low hardware overhead compared to other communication protocols, as it only requires four signal lines and doesn't require any addressing or arbitration.
6. Voltage levels: The voltage levels for the SPI protocol depend on the specific device being used, but typically range from 1.8V to 5V. It's important to ensure that the voltage levels of the SPI signals are compatible between the devices being used.
7. Overall, the SPI protocol is a flexible and versatile communication standard that can be configured to suit a wide range of devices and communication requirements. Its relatively simple and efficient design makes it a popular choice for many applications in the electronics industry [6].

3.1.2.d SPI Error detects:

The SPI protocol does not have built-in error detection or correction mechanisms, as it is a simple and efficient communication protocol. This means that it is the responsibility of the devices communicating over SPI to ensure that the data being transmitted is accurate and reliable.

However, there are a few ways that errors can be detected in SPI communication:

1. Checksum or CRC: A checksum or cyclic redundancy check (CRC) can be calculated on the data being transmitted to detect errors. The receiver can compare the calculated checksum or CRC with the one sent by the transmitter to detect any errors in the data.
2. Parity bit: A parity bit can be added to each byte of data being transmitted. The parity bit is set to either 0 or 1 to ensure that the total number of 1 bits in the byte is always even or odd. The receiver can check the parity bit to detect any errors in the data.
3. Retransmission: If an error is detected in the data, the receiver can request that the data be retransmitted by the transmitter.

3.1.2.e SPI Data frame:

In the SPI protocol, the data frame consists of a series of bits that are transmitted between the master and slave devices. The data frame typically includes the following elements:

1. Start bit: The start bit is a single bit that indicates the beginning of the data frame. It is typically set to 0.
2. Data bits: The data bits are the actual data being transmitted between the devices. The number of data bits per transfer can be configured, but is typically between 8 and 16 bits.
3. Parity bit: The parity bit is an optional bit that can be used for error detection. It is typically set to either 0 or 1 to ensure that the total number of 1 bit in the data frame is always even or odd.
4. Stop bit: The stop bit is a single bit that indicates the end of the data frame. It is typically set to 1.

The specific format of the data frame can vary depending on the configuration of the SPI protocol and the devices being used. For example, some devices may use a different number of data bits per transfer, or may not use a parity bit. It's important to consult the datasheets of the devices being used to ensure that the data frame is properly configured and aligned between the master and slave devices [6].

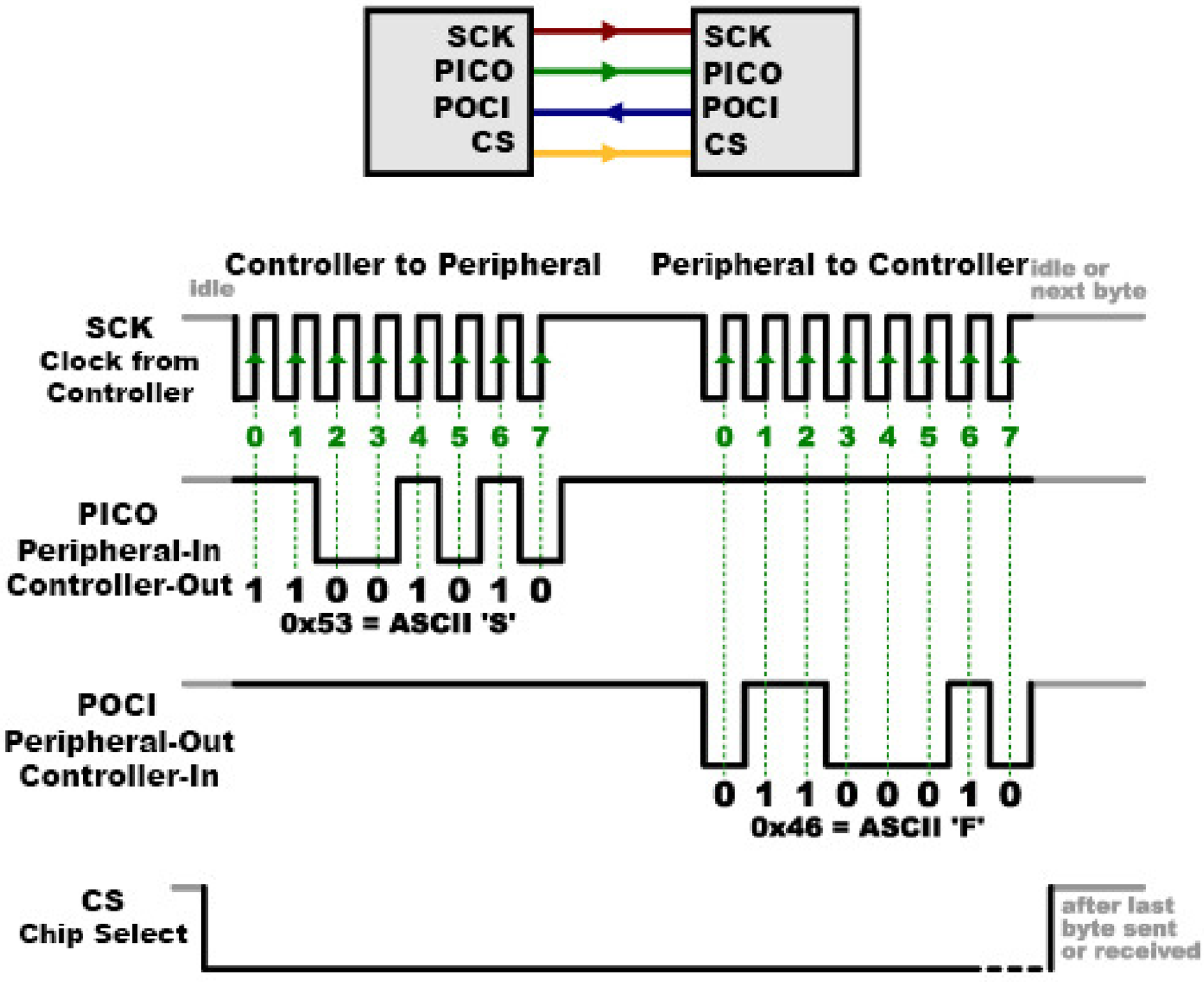


Figure 3.9 SPI data frame

3.1.2.f SPI Advantage:

The SPI protocol has several advantages, including:

1. High speed: The SPI protocol can transfer data at high speeds, making it suitable for applications that require fast communication between devices.
2. Low overhead: The SPI protocol has relatively low hardware overhead compared to other communication protocols. It only requires four signal lines and doesn't require any addressing or arbitration, making it a simple and efficient communication standard.
3. Versatility: The SPI protocol can be used to communicate with a wide range of peripheral devices, including sensors, displays, memory chips, and other microcontrollers.
4. Full-duplex communication: The SPI protocol allows for full-duplex communication, meaning that data can be sent and received simultaneously.
5. Configurable parameters: The SPI protocol has various configuration parameters, such as clock speed, clock phase and polarity, and the number of data bits per transfer. These parameters can be adjusted to optimize the data transfer rate and ensure reliable communication between the devices.
6. Low power consumption: The SPI protocol is a low-power communication standard, making it suitable for use in battery-powered devices and other low-power applications.

Overall, the SPI protocol is a reliable and efficient communication standard that is well-suited for short-distance communication between microcontrollers and peripheral devices. Its simplicity, versatility, and low overhead make it a popular choice for many applications in the electronics industry [6].

3.1.2.g SPI Usage:

The SPI protocol is a widely used communication standard in the electronics industry, and is commonly used for short-distance communication between microcontrollers and peripheral devices. Here are some of the common applications of the SPI protocol:

1. Sensor communication: Many sensors, such as temperature sensors, pressure sensors, and accelerometers, use the SPI protocol to communicate with microcontrollers. This allows the microcontroller to read data from the sensor and use it to make decisions or control other devices.
2. Memory devices: Many memory devices, such as flash memory and EEPROMs, use the SPI protocol for communication. This allows the microcontroller to read and write data to the memory device, which can be used to store program data, configuration settings, and other important information.
3. Display communication: Many displays, such as LCDs and OLEDs, use the SPI protocol for communication. This allows the microcontroller to send data to the display, which can be used to display text, graphics, and other visual information.
4. Communication between microcontrollers: The SPI protocol can also be used for communication between multiple microcontrollers. This allows the microcontrollers to exchange data and coordinate their actions, which can be useful in complex systems.
5. Communication with other peripherals: The SPI protocol can also be used to communicate with other peripheral devices, such as motor controllers, audio codecs, and wireless transceivers. This allows the microcontroller to control these devices and exchange data with them as needed

3.2 Wireless communication:

Wireless Communication is the fastest growing and most vibrant technological areas in the communication field. Wireless Communication is a method of transmitting information from one point to other, without using any connection like wires, cables or any physical medium.

Generally, in a communication system, information is transmitted from transmitter to receiver that are placed over a limited distance. With the help of Wireless Communication, the transmitter and receiver can be placed anywhere between few meters (like a T.V. Remote Control) to few thousand kilometers (Satellite Communication).

We live in a World of communication and Wireless Communication, in particular, is a key part of our lives. Some of the commonly used Wireless Communication Systems in our day – to – day life are: Mobile Phones, GPS Receivers, Remote Controls, Bluetooth Audio and Wi-Fi etc.

3.2.1 Bluetooth module:

Bluetooth wireless technology is a worldwide industry standard that enables point-to-multipoint, low-power, wireless connectivity between PCs, PDAs, mobile phones and other portable devices and connections from those devices to the LAN and Internet. The Bluetooth Special Interest Group (SIG) includes promoter companies such as 3Com, Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Nokia and Toshiba, and over 2800 other adopter/associate member companies. Bluetooth operates in the license-free 2.4GHz ISM band at a link range of 10 meters. With improved transmission power and receiving sensitivity the range can be increased up to 100 meters. Bluetooth is expected to be a standard feature on the next generation Smart Phones, PDAs and computers Bluetooth wireless technology is revolutionizing personal connectivity by providing freedom from wired connections.? Bluetooth is also revolutionizing wireless networking by providing stronger security, better resistance to interference and longer battery life than other wireless technologies. These features, in combination with the lower costs of Bluetooth technology, make it the ideal solution for handheld wireless networking [7].

With a Bluetooth handheld, you can connect to your information where you want, when you want. Whether you're managing your schedule, accessing email, or sending an instant message to a friend, Bluetooth technology allows you to take advantage of the best network available at the time. When out of the office, you can use your handheld to connect to a wide area data network through your Bluetooth-enabled mobile phone. Or you can connect at your local Bluetooth "hot spot." In the office, you can switch to Pico's Bluetooth local area network to get even faster speeds with no airtime charges. Bluetooth technology allows you to connect from anywhere and that means you can do more with your life.

3.2.1.a Bluetooth Usage:

Bluetooth can be used to control a car through a mobile application. This allows for remote control of various functions of the car, such as locking and unlocking the doors, starting the engine, adjusting the temperature, and more. The mobile application can send commands to the car via Bluetooth, which is integrated into the car's system. This technology provides a convenient and secure way for car owners to control their vehicles remotely using their mobile devices.

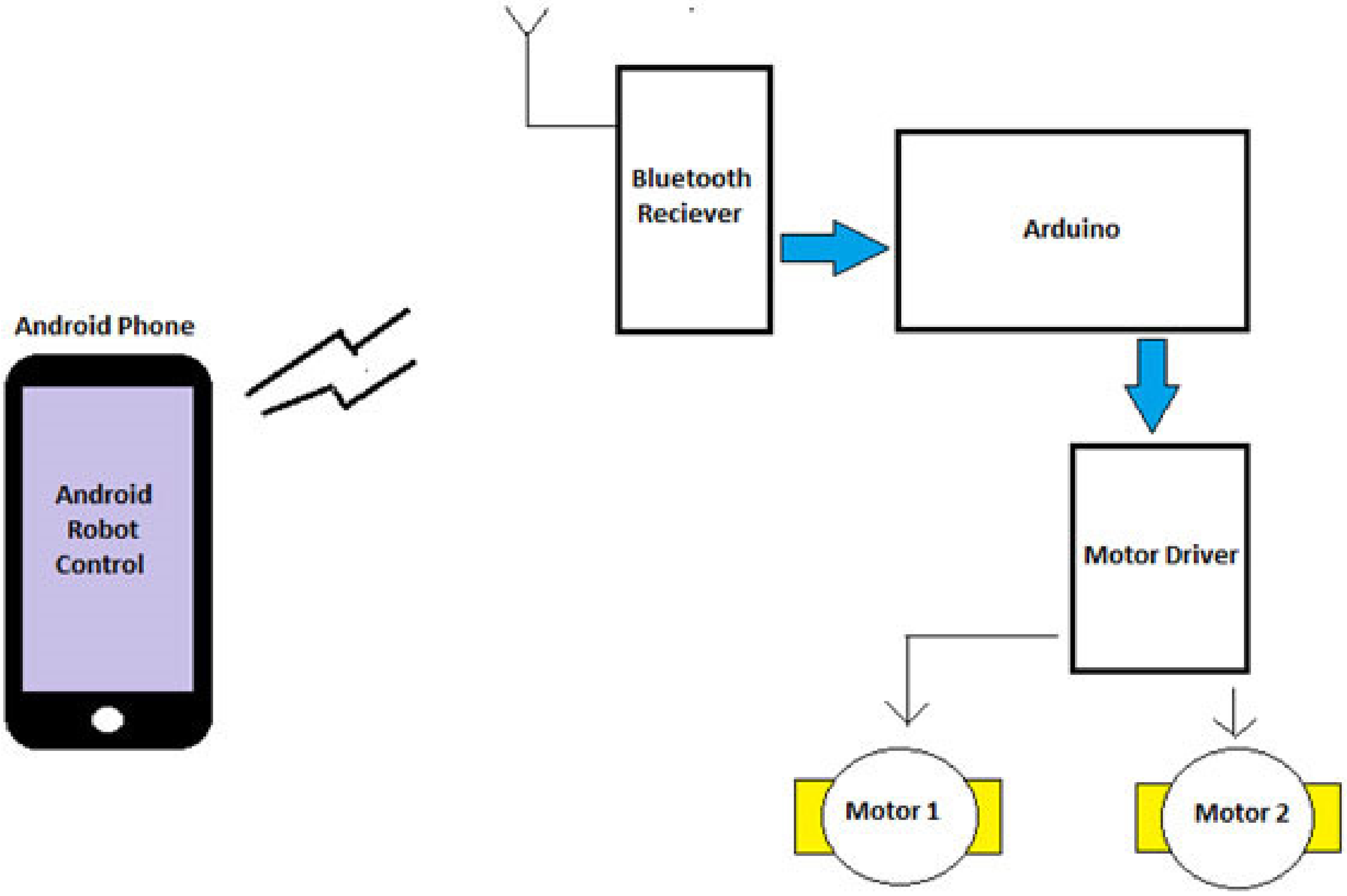


Figure 3.10 Block diagram of bluetooth subsystem

3.2.1.b How Bluetooth Works:

Bluetooth uses radio waves to transmit data between devices. It operates in the 2.4 GHz frequency band and uses a technique called frequency hopping spread spectrum (FHSS) to avoid interference with other wireless devices operating in the same frequency band.

When two Bluetooth devices are paired, they establish a secure connection with each other using a unique pairing code. Once paired, the devices can communicate with each other without the need for additional authentication. Data is transmitted between the devices in small packets, and the devices switch between transmission and reception modes to send and receive data.

Bluetooth devices can operate in two modes: master mode and slave mode. In master mode, a device initiates a connection with another device, while in slave mode, a device waits for a connection to be initiated by another device [7].

3.2.1.c Bluetooth system Blocks:

The Bluetooth system is composed of three main blocks: a radio unit, a link control unit, and a support unit for link management and host terminal interface functions (as shown in Figure 1). The Host Controller Interface (HCI) provides access to Bluetooth hardware capabilities for host devices. For example, a laptop computer could be the host device, and a PC card inserted into the PC could be the Bluetooth device. All communication between the host and the Bluetooth module goes through the HCI interface. The protocol stack resides above the radio and baseband hardware, partially in the Bluetooth unit and partially in the host device. A Bluetooth solution can also be implemented as a one-processor architecture (embedded solution), where the application and Bluetooth protocols reside in the same hardware, making the HCI unnecessary. This is a feasible implementation for simple devices such as accessories or micro servers.

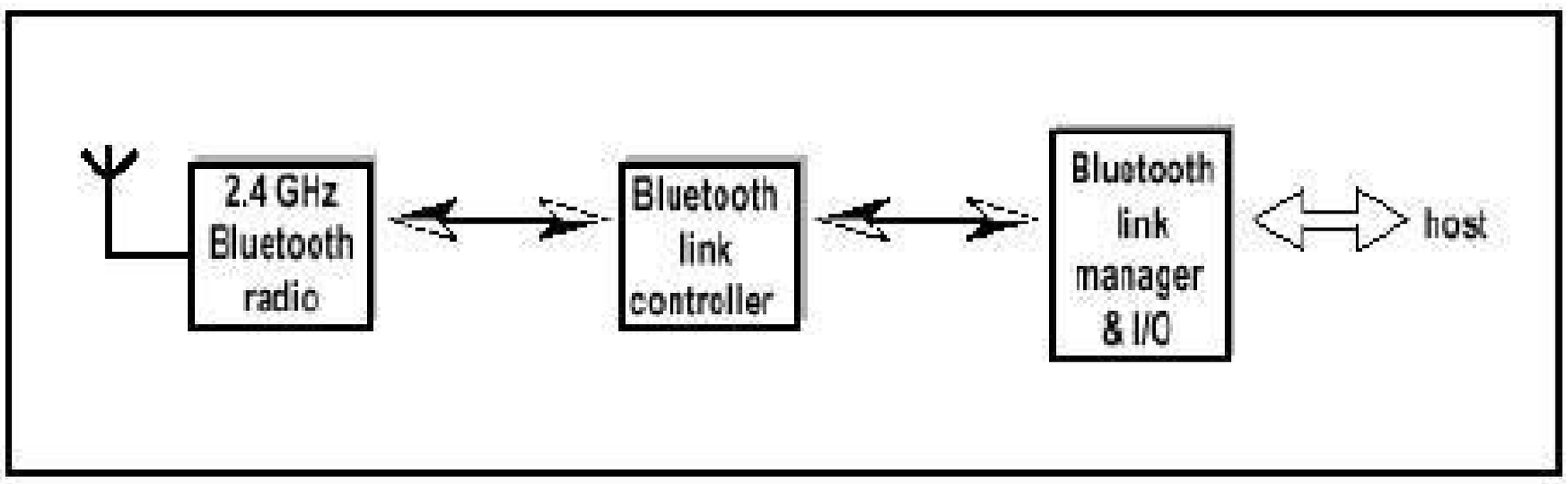


Figure 3.11 Bluetooth system Blocks

3.2.1.d Bluetooth technology:

Bluetooth technology allows for the creation of a piconet, which is the Bluetooth network. In the simplest case, a piconet consists of two connected devices, where the device that initiates the connection is called the master, and the other device is called the slave (as shown in Figure 2a). Most Bluetooth applications are point-to-point applications, with connections being ad hoc and established for a specific task, after which they are dismantled. A master device can have simultaneous connections to up to seven slaves, but the data rate is limited in this case (as shown in Figure 2b). Additionally, a device can be connected to multiple piconets, which is referred to as a scatternet (as shown in Figure 2c) [7].

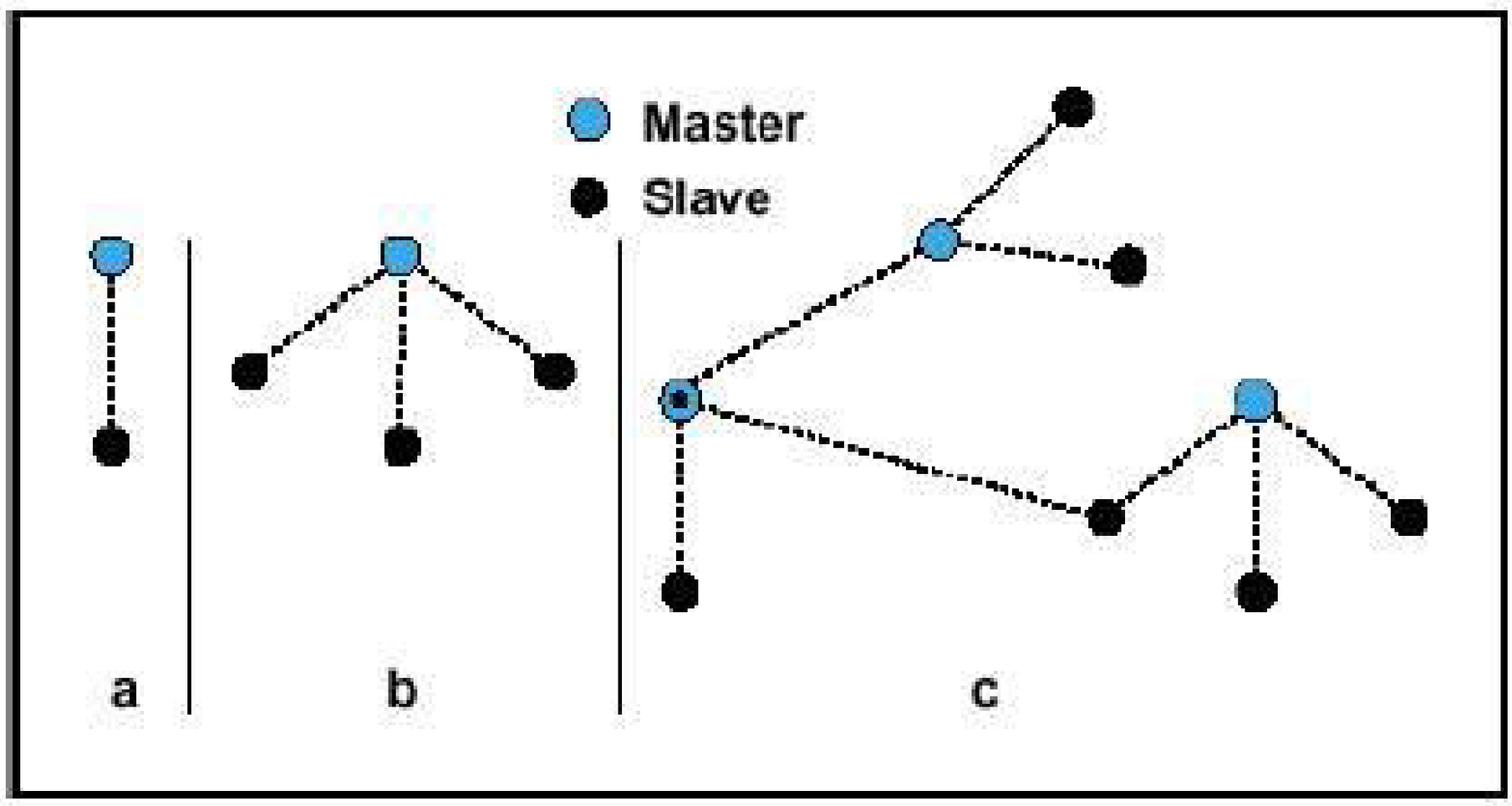


Figure 3.12 Bluetooth piconet scatternet scenarios

Bluetooth devices can communicate with each other in several ways, including:

- File Transfer: Bluetooth can be used to transfer files between devices, such as photos, music, and videos.
- Audio Streaming: Bluetooth can be used to stream audio between devices, such as playing music from a smartphone through a wireless speaker.
- Internet Connectivity: Bluetooth can be used to connect devices to the internet, such as using a smartphone as a wireless hotspot.
- Smart Home Automation: Bluetooth can be used to control smart home devices, such as turning on lights or adjusting thermostat settings.
- Driving Car: Bluetooth can be used to control cars, such as moving it Forward & Left & Backward
- And right.

3.2.1.e Bluetooth Advantages:

Bluetooth offers several advantages over other wireless communication technologies, including:

- Low Power Consumption: Bluetooth uses very little power, which makes it an ideal choice for battery-powered devices.
- Compatibility: Bluetooth is a widely adopted technology, which means that most devices are compatible with it.
- Ease of Use: Bluetooth devices can be paired and connected quickly and easily, without the need for complex setup procedures.
- Security: Bluetooth uses advanced encryption and authentication mechanisms to ensure that data transmitted between devices is secure.

3.2.1.f Bluetooth Implementation:

A typical implementation of Bluetooth involves several components, including the RF, baseband, HCI interface, and host stacks software.

The RF component operates in the license-free 2.4-2.4835GHz ISM band and uses frequency hopping at a rate of 1600 hops/s across 79 1MHz channels. The range for Bluetooth is typically 10 meters with a 1Mbps rate, but this can be increased to 100 meters with improved transmission power and receiving sensitivity.

The RF component has three classes of transmit power: Class 3 at 0 dBm (1 mW), Class 2 at 4 dBm (2.5 mW), and Class 1 at 20 dBm (100 mW). The minimum receiver sensitivity level is -70 dBm, with most products operating between -75dBm and -90dBm for a 10-3 BER.

The baseband component of Bluetooth provides cable replacement for connectivity among various devices in a wireless PAN, also known as a Piconet. Multiple devices, up to 256 parked and 8 actives, can participate in a Piconet.

Bluetooth communications in a Piconet are based on a master/slave relationship, where one unit serves as a master and the rest serve as slaves.

The access is synchronized via the master identity, whose Bluetooth address determines the frequency hopping sequence, and the system clock determines the phase.

Each slave follows the hop sequence and adds an offset to its clock to follow the master. Each Bluetooth packet has a fixed format that starts with a 72-bit access code unique to the Piconet and based on the master identity. A 54-bit header follows, containing error correction, retransmission, and control information.
Finally, a payload of 0 to 2745 bits ends a packet [7].

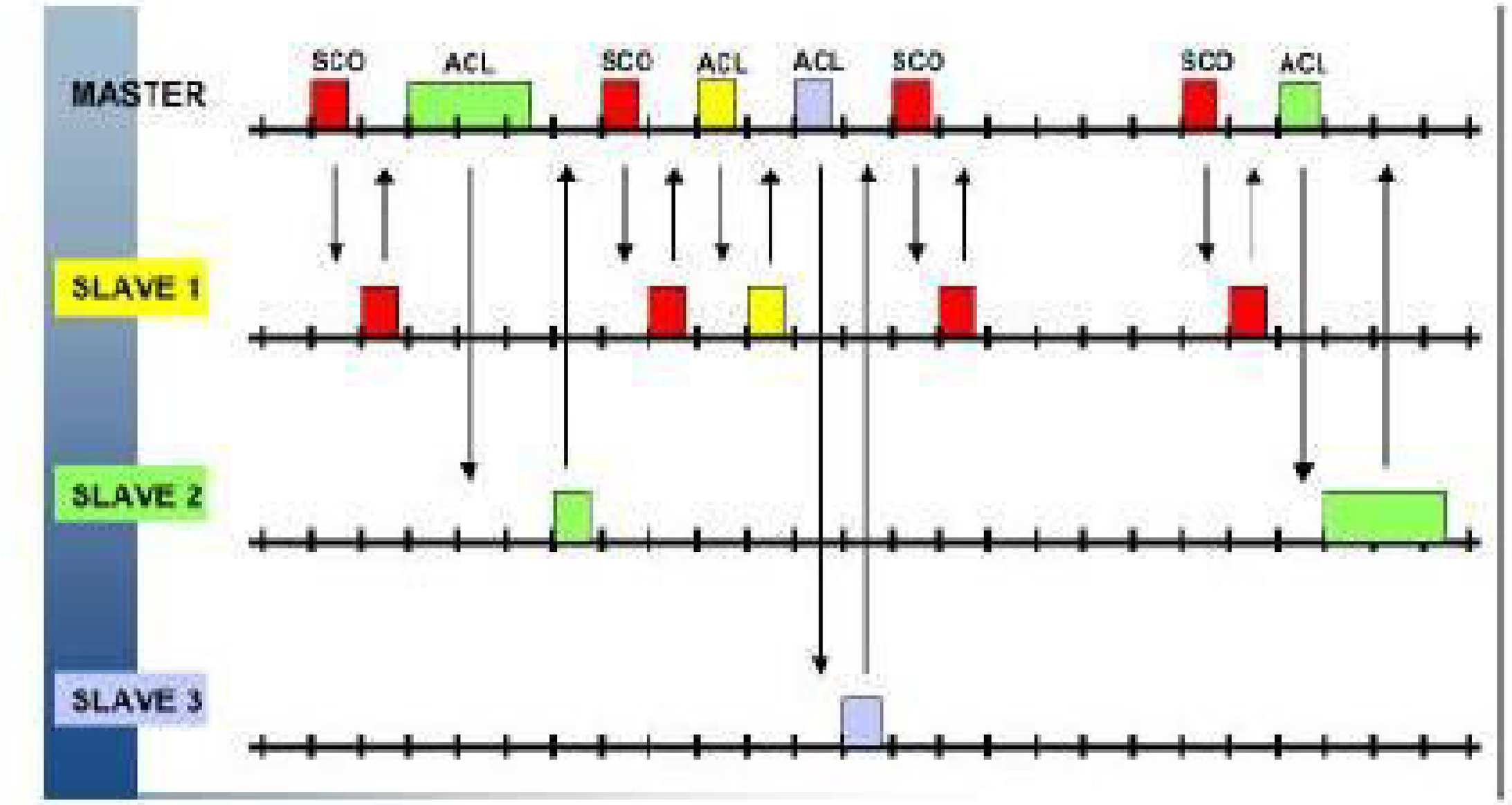


Figure 3.13 master to slave connection in a piconet

3.2.1.g Frequency hopping:

Bluetooth uses frequency hopping at a rate of 1600 hops/s, across 79 one MHz channels using a 1 bit/symbol GFSK modulation scheme. To provide full duplex operation, Bluetooth uses Time-Division Duplex (TDD) scheme to divide the channel into a number of 625 us time slots with a 220 us TDD guard time. The master and slave devices alternate transmission, with the master transmitting in even-numbered time slots, and the slave transmitting in odd-numbered time slots. Bluetooth can support three 64Kbps voice channels through synchronous connection-oriented (SCO) links for circuit-switching audio applications. It also supports asynchronous connection-less (ACL) links for packet data switching, based on a polling access scheme. The typical packet size is one slot, but it can span multiple slots as defined in the specification. Bluetooth provides a maximum data rate of asymmetrical 721 Kb/s upstream (57.6Kb/s downstream) or symmetrical 432.6 kb/s data transfer rate [8] .

3.2.2 infrared (IR) module:

Infrared (IR), also known as infrared light, is a type of electromagnetic radiation (EMR) that has longer wavelengths than visible light, making it invisible to the human eye. Typically, IR wavelengths range from around 700 nanometers, which is just beyond the red end of the visible spectrum, up to 1 millimeter (or 300 GHz), though longer wavelengths are sometimes referred to as terahertz radiation.

Almost all black-body radiation emitted by objects near room temperature falls within the IR range. As a form of EMR, IR carries both energy and momentum, exhibiting characteristics of both waves and particles, or photons.

IR was first discovered in 1800 by astronomer Sir William Herschel, who detected a type of invisible radiation below the energy level of red light by observing its effect on a thermometer. Further research revealed that slightly more than half of the total energy from the sun arrives on Earth in the form of infrared radiation. The balance between absorbed and emitted IR radiation is critical to Earth's climate [9].

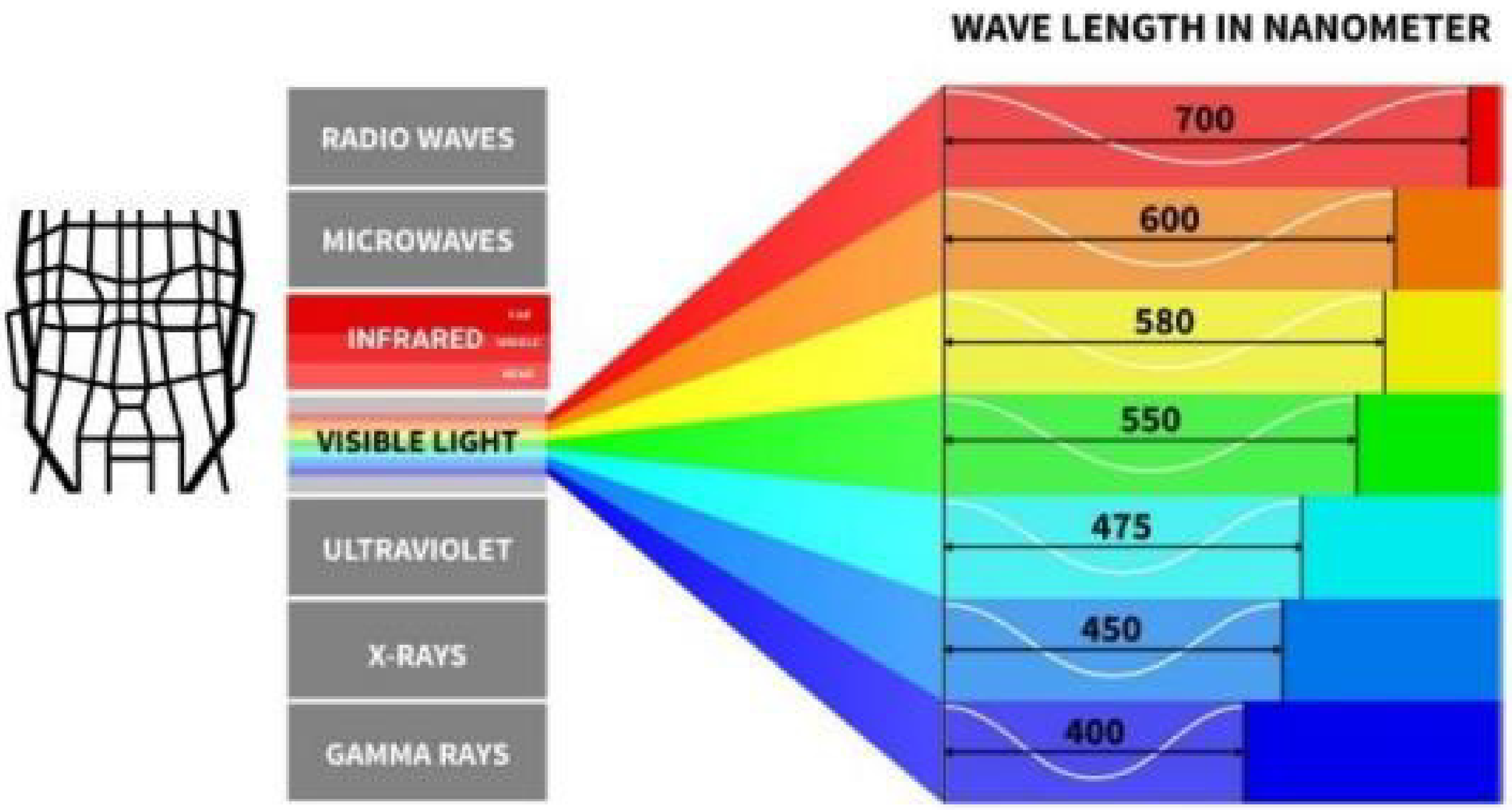


Figure 3.14 Wavelength in manometer

3.2.2.a IR Communication:

IR, or infrared, communication is a common, inexpensive, and easy to use wireless communication technology. IR light is very similar to visible light, except that it has a slightly longer wavelength. This means IR is undetectable to the human eye - perfect for wireless communication. For example, when you hit a button on your TV remote, an IR LED repeatedly turns on and off, 38,000 time a second, to transmit information (like volume or channel control) to an IR photo sensor on your TV. It has many advantages such as: 1. Safety: Infrared radiation is not harmful to human beings. Hence infrared communication can be used at any place. 2. High Speed: All of electromagnetic radiations travel at the speed of 300,000,000 meters per second. 3. Relatively Long Distance: Distance between transmitter and receiver in IR communication system is depending on the hardware and most probably ranges in meters. Most of IR systems supports data transfer on a distance up to 10 meters [9].

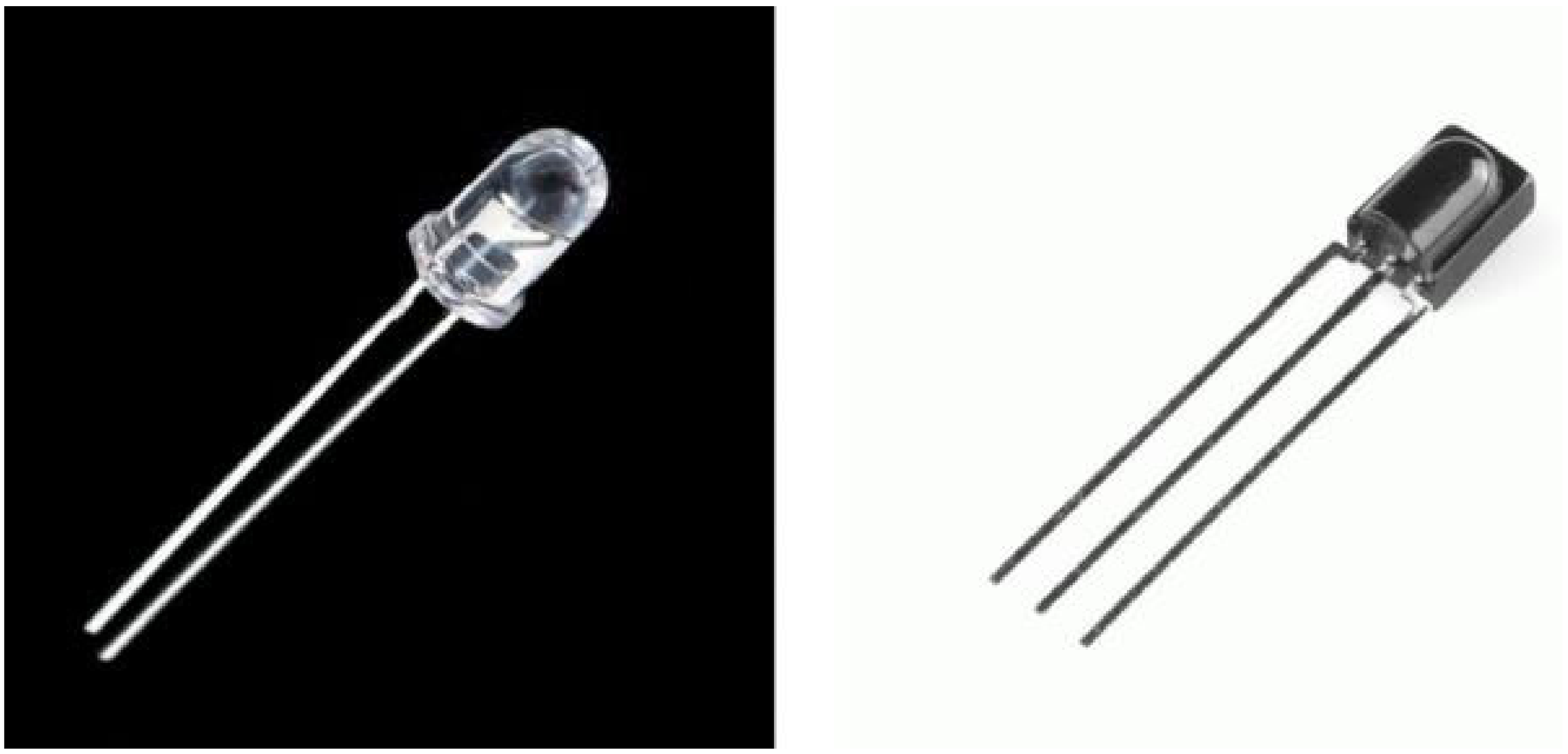


Figure 3.15 IR module

3.2.2.b IR Transmitter:

The IR transmitter is a transmitting IR LED that blinks very quickly for a fraction of time. Depending on that fraction of time, the data will be encoded either 1 or 0. For example, let's say that digital 0 is represented by a fraction of time 0.5 ms and digital 1 is represented by a fraction of time of 2 ms. Assuming the modulation is done on 38KHz frequency; then to send 0 we will blink the LED with frequency 38KHz for 0.5ms. To send 1 we will blink the LED with frequency 38KHz for 2ms [9].

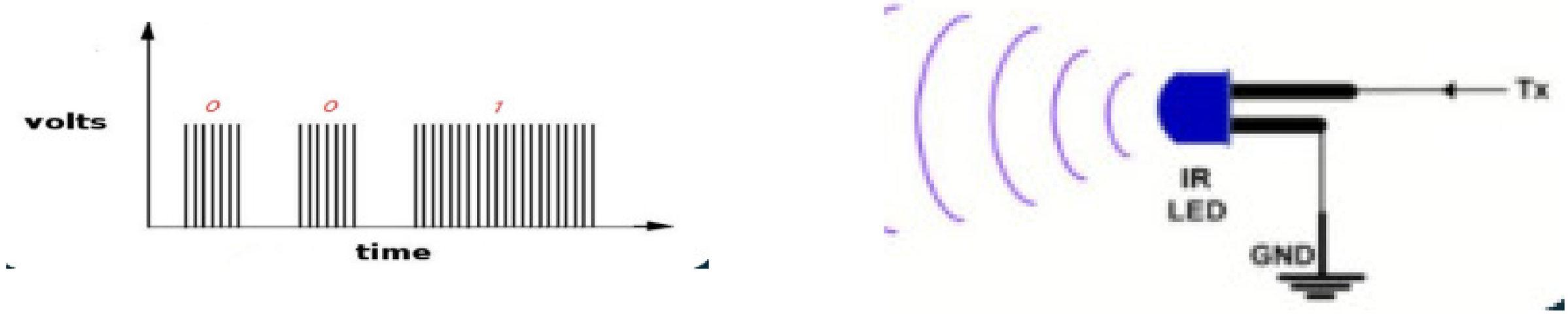


Figure 3.16 IR Transmitter

3.2.2.b IR Receiver:

Infrared receivers are also called as infrared sensors as they detect the radiation from an IR transmitter. Normally they come in a form of photodiodes or modules based on photodiodes.

Different types of IR receivers exist based on the wavelength, voltage, receiving frequency and packaging.

One of the most popular modules are based on 38KHz frequency, they have a digital output that become 0 when an IR radiation of 38 KHz is received and become 1 otherwise IR communication has many protocols identified.

These protocols are different mainly in the encoding and decoding scheme. One of the most popular protocol is NEC code.

Other protocols are exist such as Toshiba Micom Format, Sharp Code, RC5 Code, RC6 Code, R–2000 Code, Sony Format. In our reference we will focus on NEC protocol [9].

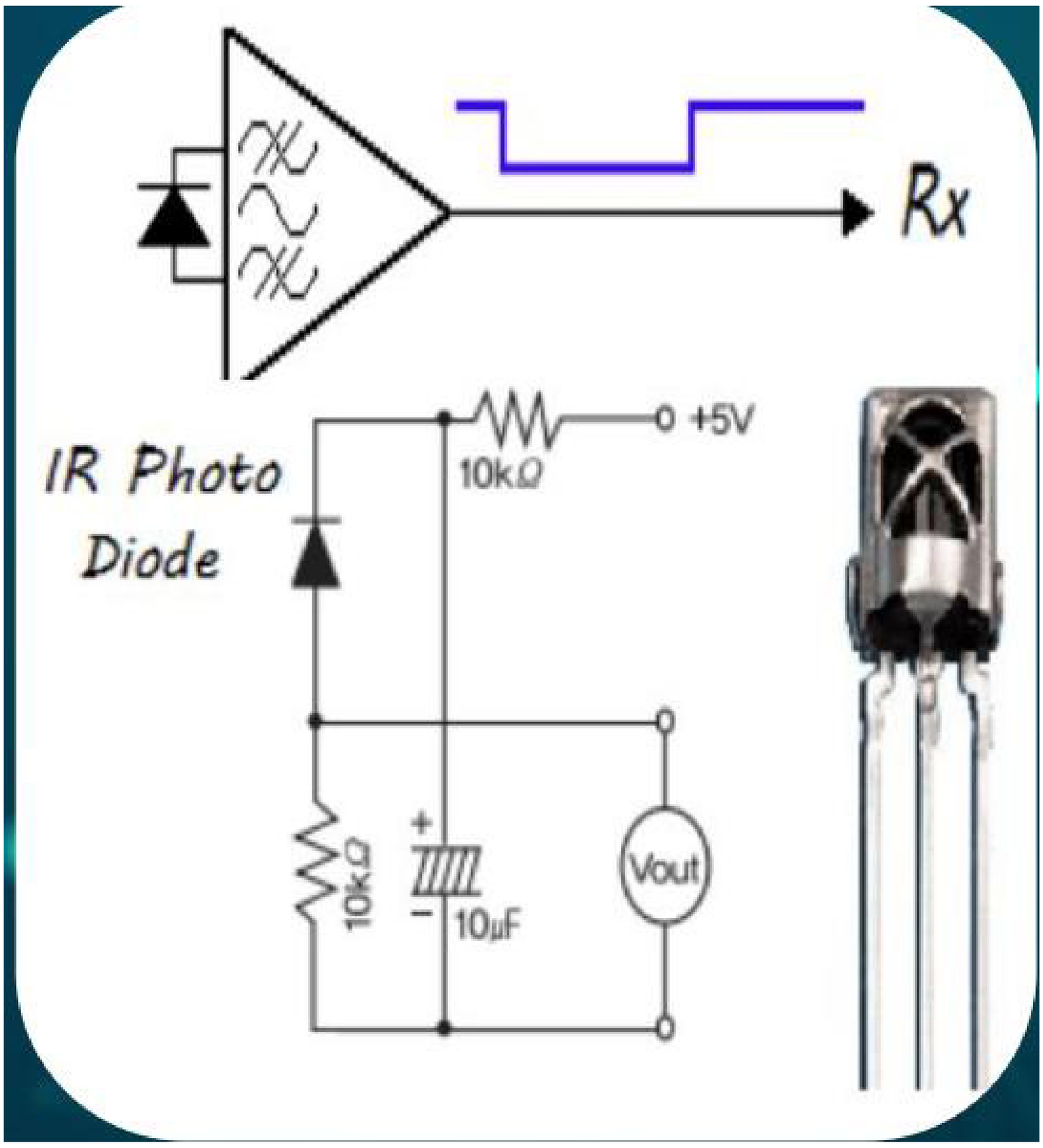


Figure 3.17 IR Receiver

3.2.2.c Nec Protocol:

When a key is pressed on the remote controller, the message transmitted consists of the following, in order: 9ms leading pulse burst 4.5ms space

- 8-bit address for the receiving device
- 8-bit logical inverse of the address
- 8-bit command
- 8-bit logical inverse of the command
- 562.5µs pulse burst to signify the end of message transmission.

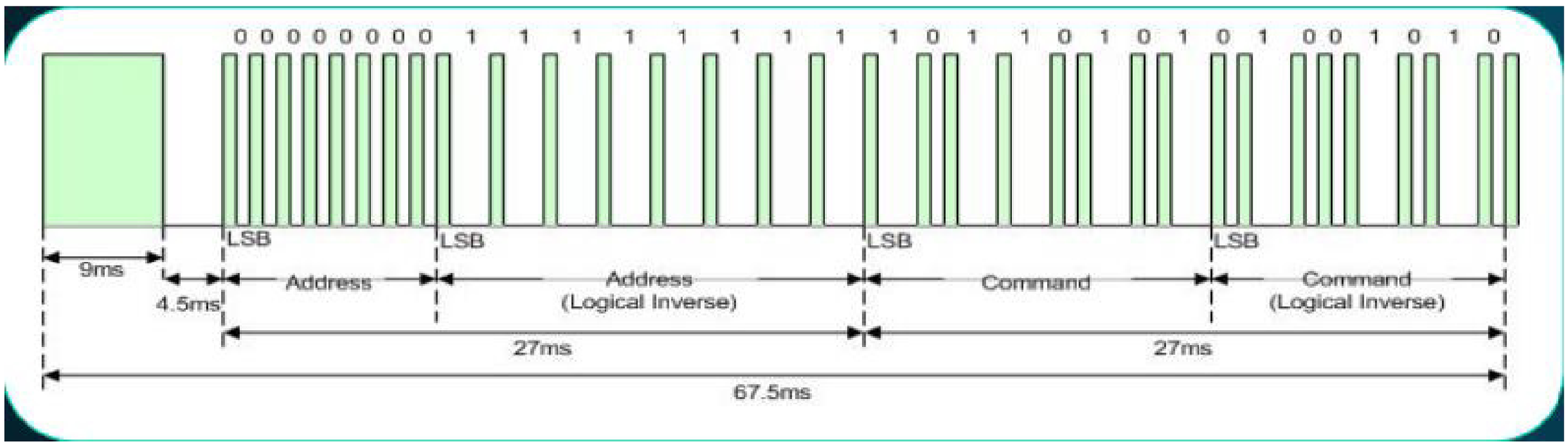


Figure 3.18 Nec frame

3.2.3 WIFI module:

Wi-Fi is a widely used technology, with most homes, businesses, and public places offering some form of Wi-Fi connectivity. It has become an essential part of modern life, enabling people to stay connected and access information on the go. Wi-Fi also enables the Internet of Things (IoT), allowing devices such as smart appliances, security systems, and wearable technology to connect and communicate with each other.

3.2.3.1 Raspberry pi and WIFI:

One of the key features of the Raspberry Pi is its ability to connect to wireless networks using WiFi. In this way, the Raspberry Pi can be used as a low-cost and flexible platform for a wide range of wireless applications.

3.2.3.1.a Why use WIFI to communicate between raspberry pis:

Wi-Fi is often used to communicate between two Raspberry Pis because it provides a convenient and flexible way to connect the devices without the need for physical cables. With Wi-Fi, the two Raspberry Pis can communicate wirelessly, which can be especially useful in situations where running cables between the devices is not practical or feasible.

In addition to its convenience, Wi-Fi also offers a relatively high data transfer rate, allowing for efficient data exchange between the Raspberry Pis. This can be important in applications where real-time or near real-time communication is required.

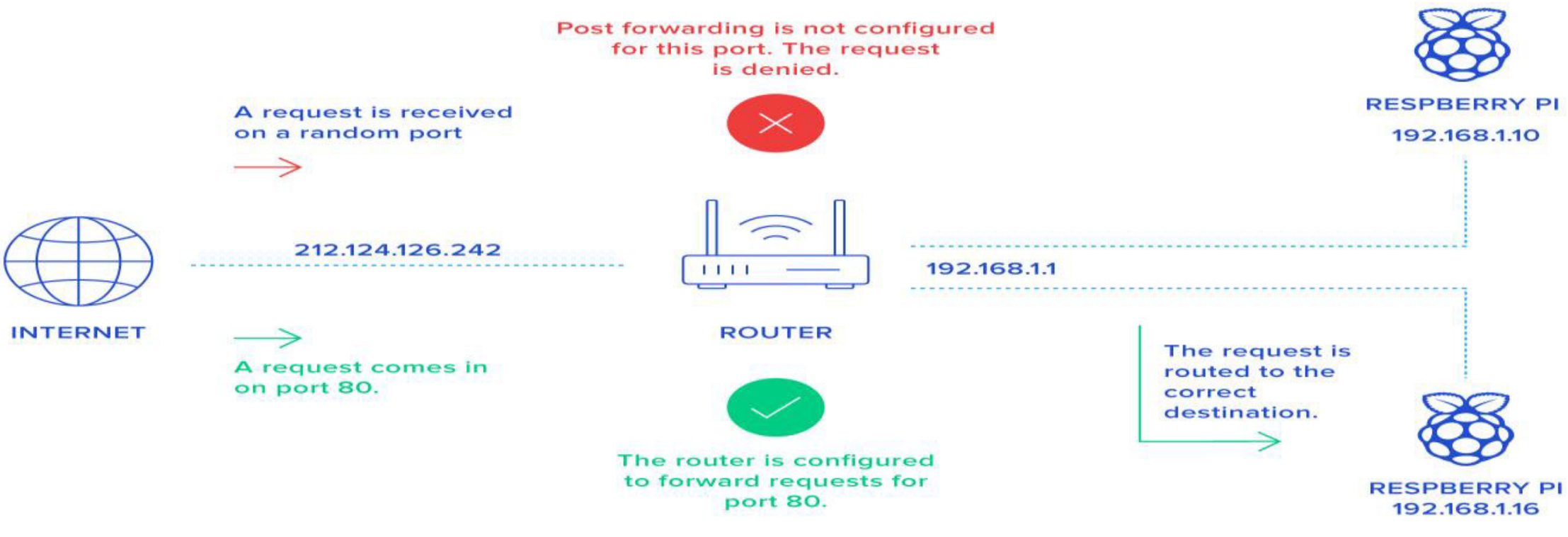

Post forwarding is not configured for this port. The request is denied.
A request is received on a random port
RESPBERRY PI
192.168.1.10
212.124.126.242
192.168.1.1
INTERNET
ROUTER
A request comes in on port 80.
The request is routed to the correct destination.
The router is configured to forward requests for port 80.
RESPBERRY PI
192.168.1.16

Figure 3.19 WIFI communication between two raspberry pis

3.2.3.1.b Setting up your raspberry pis:

To use WiFi on a Raspberry Pi, you will need a WiFi dongle or a Raspberry Pi model with built-in WiFi. Once you have the necessary hardware, you can configure the Raspberry Pi to connect to a WiFi network using the command line or a graphical user interface.

3.2.3.2 Socket programming:

In the 1980s, the US government's Advanced Research Projects Agency (ARPA) provided funds to the University of California at Berkeley to implement TCP/IP protocols under the UNIX operating system. During this project, a group of Berkeley researchers developed an application program interface (API) for TCP/IP network communications called the socket interface

3.2.3.2.a What is socket programming:

Socket programming is a way of enabling communication between two devices over a network using sockets. A socket is a software endpoint that allows two devices to communicate with each other over a network.

In socket programming, one device acts as a server, and the other device acts as a client. The server creates a socket and waits for a client to connect to it. The client then connects to the server's socket, and the two devices can exchange data through the socket.

Sockets: uniquely identified by:

- an internet address
- an end-to-end protocol (e.g. TCP or UDP)
- a port number [10]

3.2.3.2.b Client-server communication:

A network is composed of computers which is either a client or a server. A server is a program that is offering some service whereas a client is a program that is requesting some service.

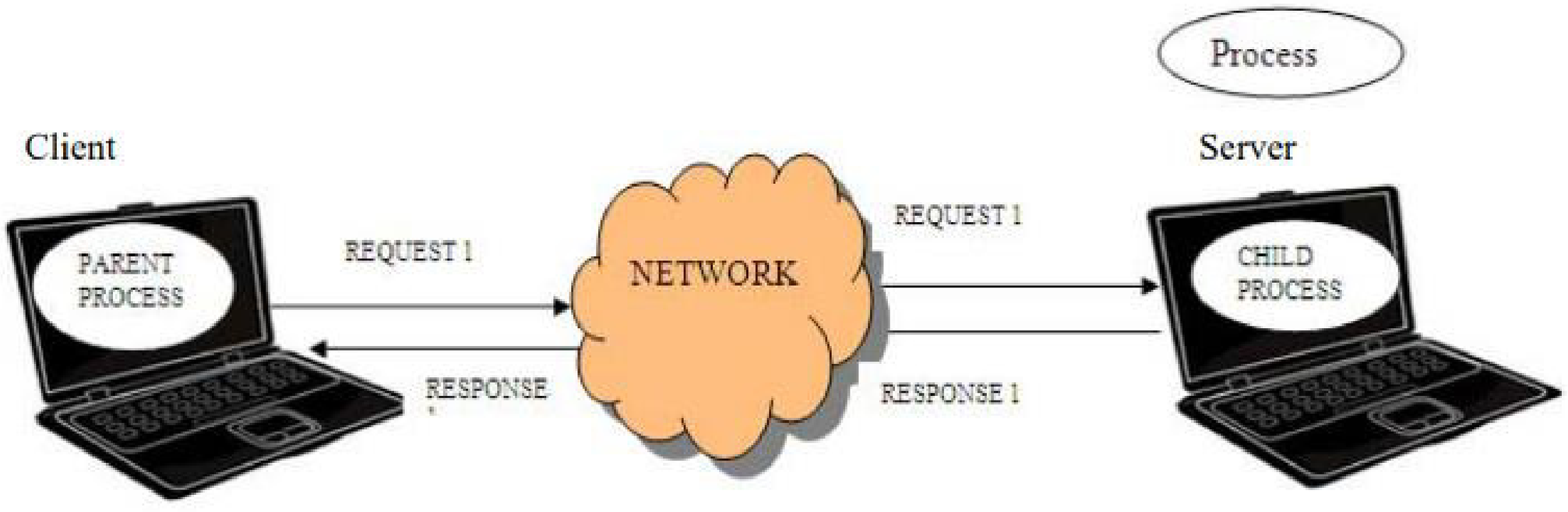


Figure 3.20 client-server communications

3.2.3.3 Methodology:

In this project two Raspberry Pis communicated to each other VIA WIFI using socket, one Raspberry Pi is used as the server and the other as the client, in which the server sends the data to the client.

3.2.3.3.a General steps:

- Set up Wi-Fi on each Raspberry Pi: You will need to configure the Wi-Fi settings on each Raspberry Pi so that they are connected to the same Wi-Fi network or create a Wi-Fi ad-hoc network between them. You can do this by using the Raspberry Pi's graphical user interface or by modifying the network configuration files.
- Find the IP addresses of each Raspberry Pi: Once the Raspberry Pis are connected to the same Wi-Fi network, you will need to find their IP addresses. You can do this by using the ifconfig command in the terminal or by using a network scanner tool.

The hostname –I command is used to display the Raspberry Pi's current IP address[12]

- Install and configure any necessary software: Depending on what you want to do with the Raspberry Pis, you may need to install and configure additional software. For example, if you want to transfer files between the Raspberry Pis, you can install and configure Samba or SCP.
- Test the communication: Once you have set up Wi-Fi and installed any necessary software, you can test the communication between the Raspberry Pis. You can do this by pinging the IP address of the other Raspberry Pi or by using any software you installed to transfer files or data.

3.2.3.3.b Hardware list:

- Raspberry Pi 3 B+: Small and portable board that can perform as a computer. It acts as a file server in this work.
- SD card: Raspbian OS in SD card
- Wireless router: Provide wireless environment for file access between client and server.

Figure 3.21 Raspberry Pi 3 B+

Figure 3.22 SD card

Figure 3.23 Wireless router

3.2.3.3.c Software list:

- Raspbian OS: It is the official image for RaspberryPi, software development for this OS includes python language
- Python3: for code implementation

3.2.3.4 Client-Server Communication within Localhost:

In this scenario, python script is written for communication between client and server within localhost [11].

One Raspberry Pi which is the server is used to send data to the client which is the other RaspberryPi

3.2.3.4.a Coding:

For this section the python script for both the server and the client is shown :

- ***Server code***

```
import serial
import socket
import time
SERVER_ADDRESS = '172.20.10.2'
SERVER_PORT = 5560
# Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Bind the socket to the server address and port
server_socket.bind((SERVER_ADDRESS, SERVER_PORT))
# Listen for incoming connections
server_socket.listen()
# Wait for a client to connect
print('Waiting for a client connection...')
client_socket, client_address = server_socket.accept()
print(f 'Client connected: {client_address}')
ser = serial.Serial('/dev/ttyS0', 9600, timeout=1)
ser.reset_input_buffer()
while True:
        ser.reset_input_buffer()
        data = ser.read()
        #Data=data.decode()
        print(data)
        #Data=1
```

```
        try:
            client_socket.sendall(data)
            print(f 'Data sent to client: {data}')
        except BrokenPipeError:
            print('Client disconnected')
            break
        time.sleep(1)
client_socket.close()
server_socket.close()
```

In this code the IP address of the server is assigned to **SERVER_ADDRESS** with the port to **SERVER_PORT**. After that **socket.socket(socket.AF_INET, socket.SOCK_STREAM)** is used to create a socket object and then the socket was bound to the server address and the port using **server_socket.bind((SERVER_ADDRESS, SERVER_PORT)).** Following that it's time for the server to listen for incoming connections and connect to it **server_socket.listen()**

```
# Wait for a client to connect
print('Waiting for a client connection...')
client_socket, client_address = server_socket.accept()
```

.

After connecting to the client the required data is sent to the client VIA **client_socket.sendall(data)** , unless there is a broken pipe, it won't break from the connection.

- ***Server code***

```
import socket
import time
# Define the server address and port
SERVER_ADDRESS = '172.20.10.2'
# Create a socket object
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect to the server
client_socket.connect((SERVER_ADDRESS, SERVER_PORT))
# Receive data from the server every one second
while True:
    try:
        data = client_socket.recv(1024)
        if not data:
            print('Server disconnected')
            break
        print(f 'Response from server: {ord(data.decode())}')
    except ConnectionResetError:
        print('Server disconnected')
        break
    #Wait for one second before receiving the next data packet
    time.sleep(1)
# Close the socket
client_socket.close()
```

Firstly, we define the server's address and port. Accordingly **client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)** is used to create a socket object, after that **client_socket.connect((SERVER_ADDRESS, SERVER_PORT))** is used to connect with the server, finally the client tries to read the sent data from the server **client_socket.recv(1024)**.

Embedded Linux Subsystem

4

In this chapter, we will talk about Linux and its features. Also, will talk about Embedded Linux, its Architecture, and explain its component individually. Finally, we will explain what the Build systems is and explain Yocto in detail.

4.1 Introduction:

Nowadays, Technology development increases rapidly, so we need applications more complicated with a huge number of features. To deal with these expectations we need powerful tools, like the Linux operating system to control these complex applications of embedded systems devices. Ten years ago, embedded processors had limitations and didn't have the same capabilities as today's processors. Now microcontrollers or system on chip (SOC) generally provide interfaces with high application and suitable memory size and high speed. undoubtedly, Embedded Linux will be applied to more and more devices. Linux provides the flexibility and power of open-source development and offers greater security. also, in a bare-metal environment, we are limited to a single application image. As we build out the application, we‟ll notice things get hard if our system must do a few totally different things simultaneously. If we‟re developing for Linux, we can break this functionality into separate processes, where we can develop, debug, and deploy separately as separate binary images. So, let's know more details about Linux [13].

4.2 Linux:

Linux is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work [14].

4.2.1 Linux History:

In the late 1960s, Bell Labs was involved in a project with MIT and General Electric to develop a time-sharing system, called Multiplexed Information and Computing Service (Multics). Multics doesn‟t make a good success, so Bell labs decided to pull out [42].

In 1969, a team of programmers in Bell Labs who worked in Multics decided to make a better OS, they called it Uniplexed Information and Computing Service (UNICS). The team was led by great programmers:

•Ken Thompson • Dennis Ritchie

In 1971, the First Edition of Unics was released. In 1973, Ritchie rewrote B and called the new language the C language. In 1975, Unix V6 became very popular. Unix V6 was free and was distributed with its source code to many universities. Hence, many organizations had modified the Unix source code to build their own distributions. The most popular distributions are:

- Berkeley Software Distribution (BSD)

In 1983, Bell Labs found that there are many distributions of Unix that are uncontrolled and untracked. They decided to make a closed commercial version of Unix and called it System V. In 1983, Richard Stallman started the development of 100% free software. Not 90% and not 99.5%. It is totally free. Free not mean free in cost, but free from freedom! It means that this system would give the user:

- The freedom to run the program as you wish
- The freedom to copy the program and give it
- away to your friends and co-workers
- The freedom to change the program as you wish
- The freedom to distribute an improved version and thus help build the community.

GNU is a recursive acronym meaning GNU's Not Unix, A Unix-like operating system includes a kernel, compilers, editors, text formatters, mail software, graphical interfaces, libraries, games, and many other things. Thus, writing a whole operating system is a very large job. Stallman started in January 1984. One of these most important components is the GNU C Compiler GCC. By 1990 we had either found or written all the major components except one the kernel. In 1991, Linus Torvalds build a free, open-source kernel name it Linux.

Combining Linux with the almost-complete GNU system resulted in a complete operating system: GNU/Linux system[14].

Estimates are that tens of millions of people now use GNU/Linux systems. Nowadays, there are many distributions of GNU/Linux systems such as:

Ubuntu Gentoo CentOS Debian Linuxmint Fedora

4.2.2 Why Linux:

Linux is like other operating systems such as Windows, macOS, or IOS. Like them, Linux can have a graphical interface and the types of desktop software that you are used to, such as word processors, photo editors, video editors, etc. But Linux also differs in many important ways. First, and perhaps its most important feature, it is open-source software. The code used to create Linux is free and available for the public to view, edit, and for users to contribute to it. Although the core pieces of the Linux operating system are generally widespread, many Linux distributions include different software options. That means that Linux is incredibly customizable. Linux is present in the software of many devices that we use daily. Linux is secure, flexible, and can receive excellent support from a large community of users. Linux is Multitasking and Multiuser. In Linux, you get superuser access and privileges (for real), as Linux OS will not take any step without the consent of the superuser. While Linux installation, you don‟t need to install drivers for Wi-Fi, Bluetooth, mouse, touchpad, etc. explicitly as they can be installed during installation with little patience. Linux community is loyal to all the Linux users so it would give Long Term Support. Linux covers many device drivers. Linux is ported on different platforms and is platform-independent. For other OSs, usually, a user would have to go to the manufacturer‟s website to get driver support for different types of hardware. The Linux kernel supports most of the hardware automatically via plug˘and-play (largely in part because of the open-source community).

Some manufacturers also develop Linux versions of their proprietary drivers which could be easily installed via the software repository of distribution or by manually installing the provided binaries. For these and other reasons, we are seeing an accelerated adoption rate of Linux in many Applications [14].

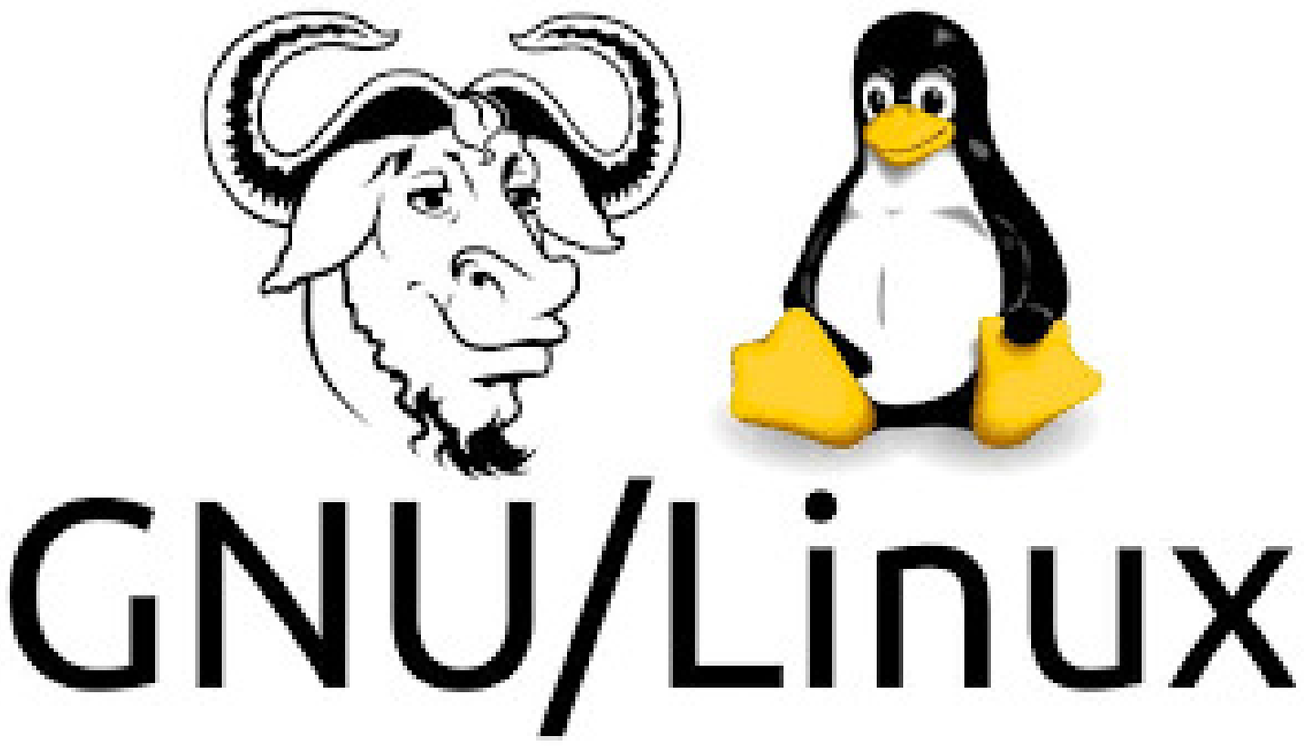


Figure 4.1 GNU/Linux OS

4.2.3 The difference between Linux an Windows:

Key	Linux	Windows
Open Source	Linux is Open Source and is free to use.	Windows is not open source and is not free to use.
Case sensitivity	Linux file system is caseˇ sensitive.	Windows file system is case insensitive.
kernel type	Linux uses a monolithic kernel.	Windows uses a microkernel.
Efficiency	Linux is more efficient in operations as compared to Windows.	Windows is less efficient in operations.
Path Separator	Linux uses a forward slash as a path separator between directories.	Windows uses backward slash as a path separator.
Security	Linux is highly secure as compared to Windows.	Windows provides less security as compared to Linux.

Table 4.1 Linux vs Windows

4.3 Embedded Linux:

Embedded Linux is a type of Linux operating system/kernel that was designed to be installed and used in embedded devices or systems. An embedded system is a set of computer hardware and software based on a microcontroller or microprocessor, controlled by a real-time operating system or RTOS, with limited memory, and that can vary both in size and complexity. Although it uses the same kernel, embedded Linux is quite different from the standard operating system. First, it gets Customized for embedded systems and, therefore, is much smaller in size, requires less processing power, and has minimal features. The Linux kernel is modified and optimized as an embedded Linux version. Such a Linux instance can only run applications created specifically for the device [15].

4.4 Embedded Linux Architecture:

Every Embedded Linux project begins by obtaining, customizing, and deploying these Elements(Components):

- Toolchain
- Bootloader
- Kernel
- Root filesystem
- Specific Application

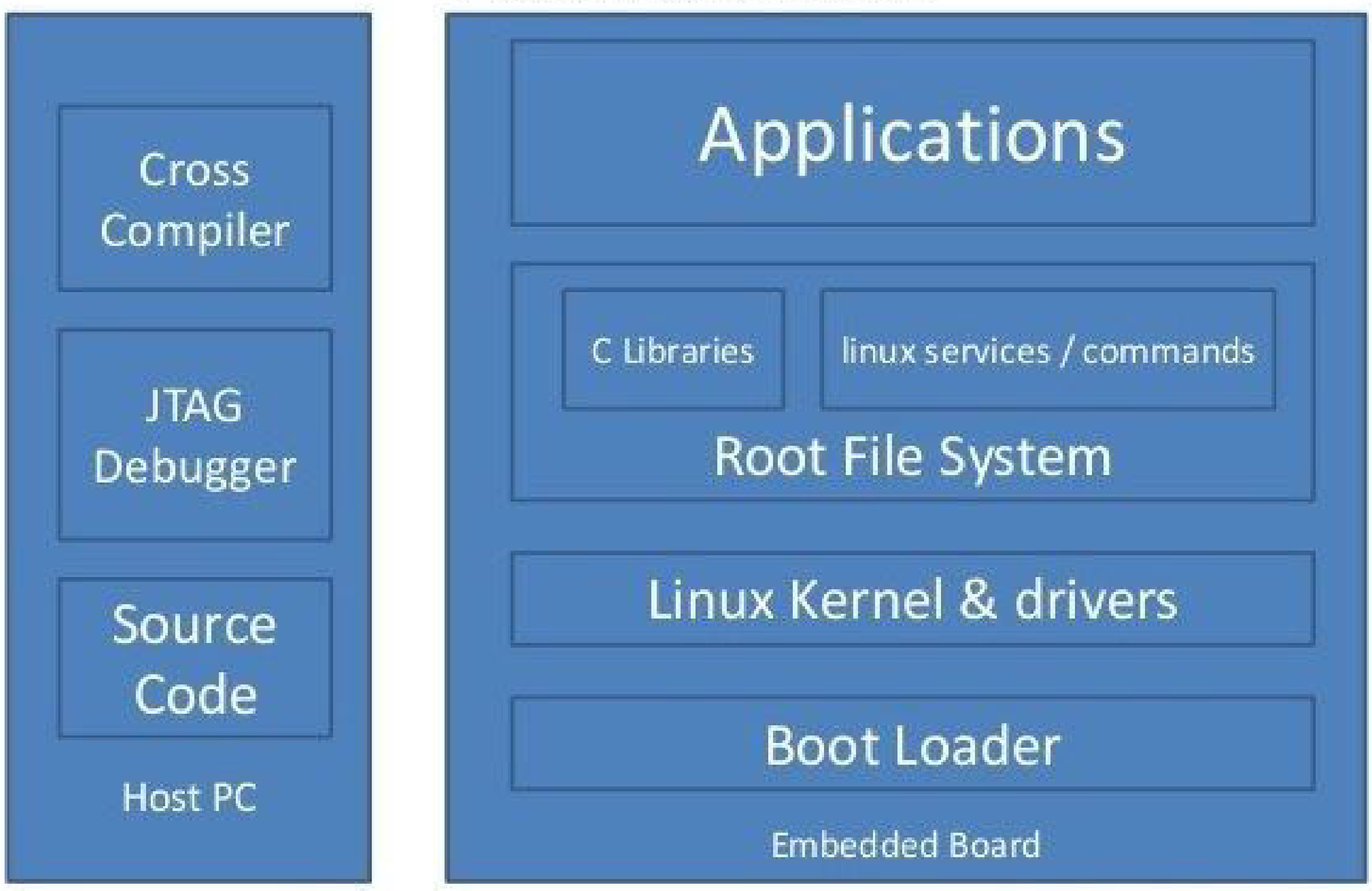


Figure 4.2 Embedded Linux Architecture

The embedded system build process is usually done on the host PC using cross˘compilation tools. Because target hardware does not have enough resources to run tools that are used to generate a binary image for target embedded hardware. The process of compiling code on one system (host system) and generating source code runs on the other system is known as cross-compilation [16].

4.4.1 Toolchain:

A toolchain is a set of tools that compiles source code into executables that can run on the target device and includes a compiler, a linker, and run-time libraries. Initially, need one to build the other three elements of an embedded Linux system: the bootloader, the kernel, and the root filesystem. It has to be able to compile code written in assembly, C, and C ++ since these are the languages used in the base open source packages [16].

There are two types of toolchain:

- Native toolchain: This toolchain runs on the same type of system, sometimes the same actual system, as the programs it generates. This is the usual case for desktops and servers.
- Cross toolchain: This toolchain runs on a different type of system than the target, allowing the development to be done on a fast desktop PC and then loaded onto the embedded target for testing. Almost all embedded Linux development is done using a cross-development toolchain, partly because most embedded devices are not well suited to program development since they lack computing power, memory, and storage, but also because it keeps the host and target environments separate [17].

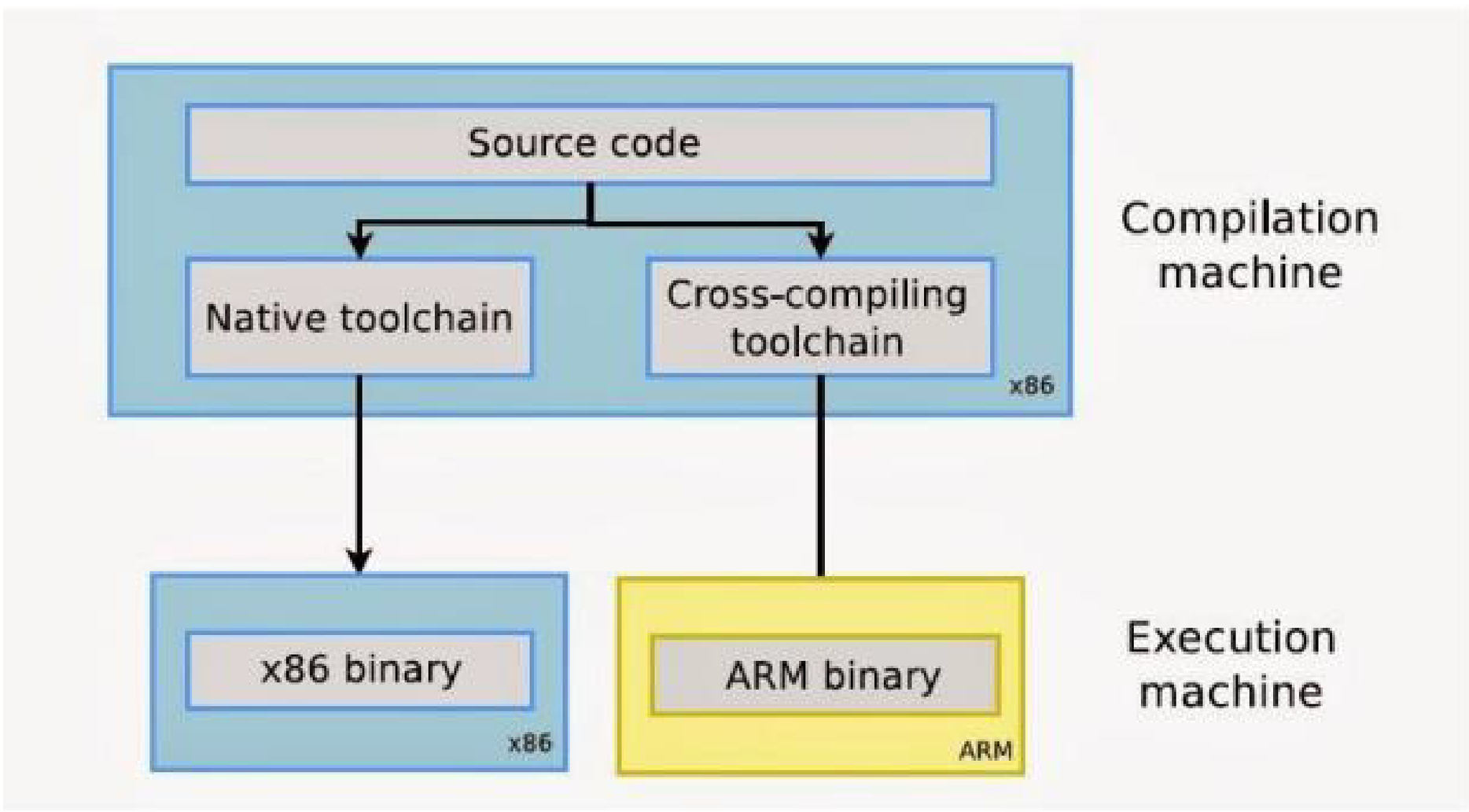


Figure 4.3 Types of Toolchain

Toolchain consists of:

- **Binutils:** is a set of tools to generate and manipulate binaries for a given CPU architecture. ex: as, ld, ar, and ranlib.

- **Kernel headers:** The C library and compiled programs need to interact with the kernel. For example, to get available system calls and their numbers or for available data structures. That‟s why compiling the C library requires kernel headers, and many applications also require them.

- **GCC:** GNU Compiler Collection, the famous free software compiler. Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Black n, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86 64, IA64, and Xtensa.

- **C library:** The C library is a very important component of a Linux system Interface between the applications and the kernel. It provides the well-known standard C API to ease application development. Several C libraries are available: glibc, uClibc, eglibc, dietlibc, newlib, etc. The choice of the C library for our toolchain must be made at the time of the cross-compiling toolchain generation.

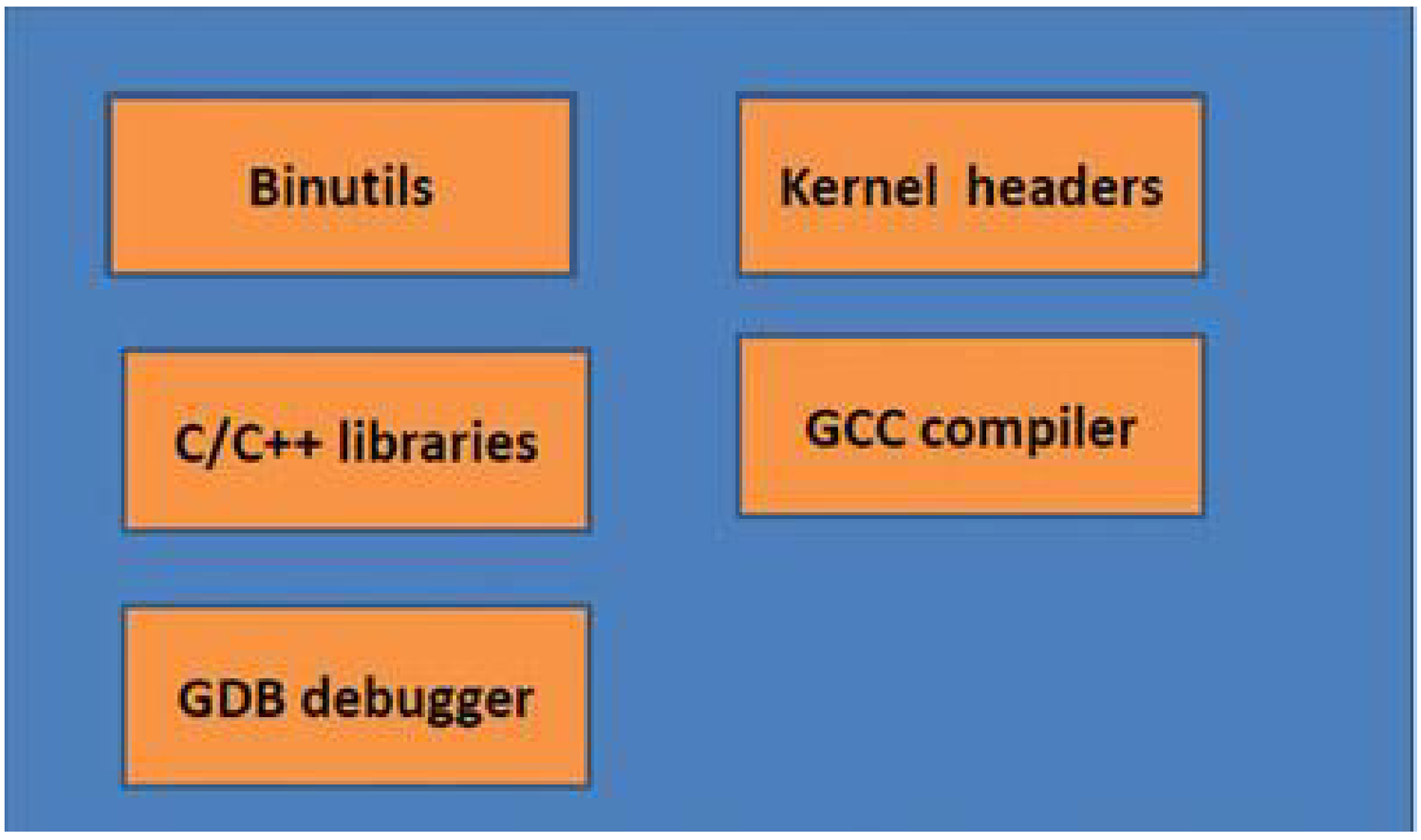


Figure 4.4 Cross-compilation toolchain

4.4.2 Bootloader:

The bootloader has two main jobs: basic system initialization and the loading of the kernel. In fact, the first job is somewhat subsidiary to the second in that it is only necessary to get as much of the system working as is needed to load the kernel. When the computer is powered on, after performing some initial setup, it will load a bootloader into memory and run that code. The bootloader‟s main job is to find the operating system‟s binary program, load that binary into memory, and run the operating system. In our case, this is the Linux kernel. The bootloader is done at this point, and all of its code and data in RAM are usually overwritten by the operating system. The bootloader won‟t run again until the computer is reset or power cycled again [17].

The bootloader in embedded systems is different from a typical laptop, desktop, or server computer. A typical PC usually boots into what we call the BIOS first and then runs Grub as the bootloader. Embedded Linux systems boot using Das-UBoot or UˇBoot for short as the bootloader. So, Let's Talk about boot sequence firstly for Linux machines in general then for Raspberry pi.

Linux machine booting sequence: the time between system power on and login is called the booting time, there are 6 high-level stages of a typical Linux boot process.

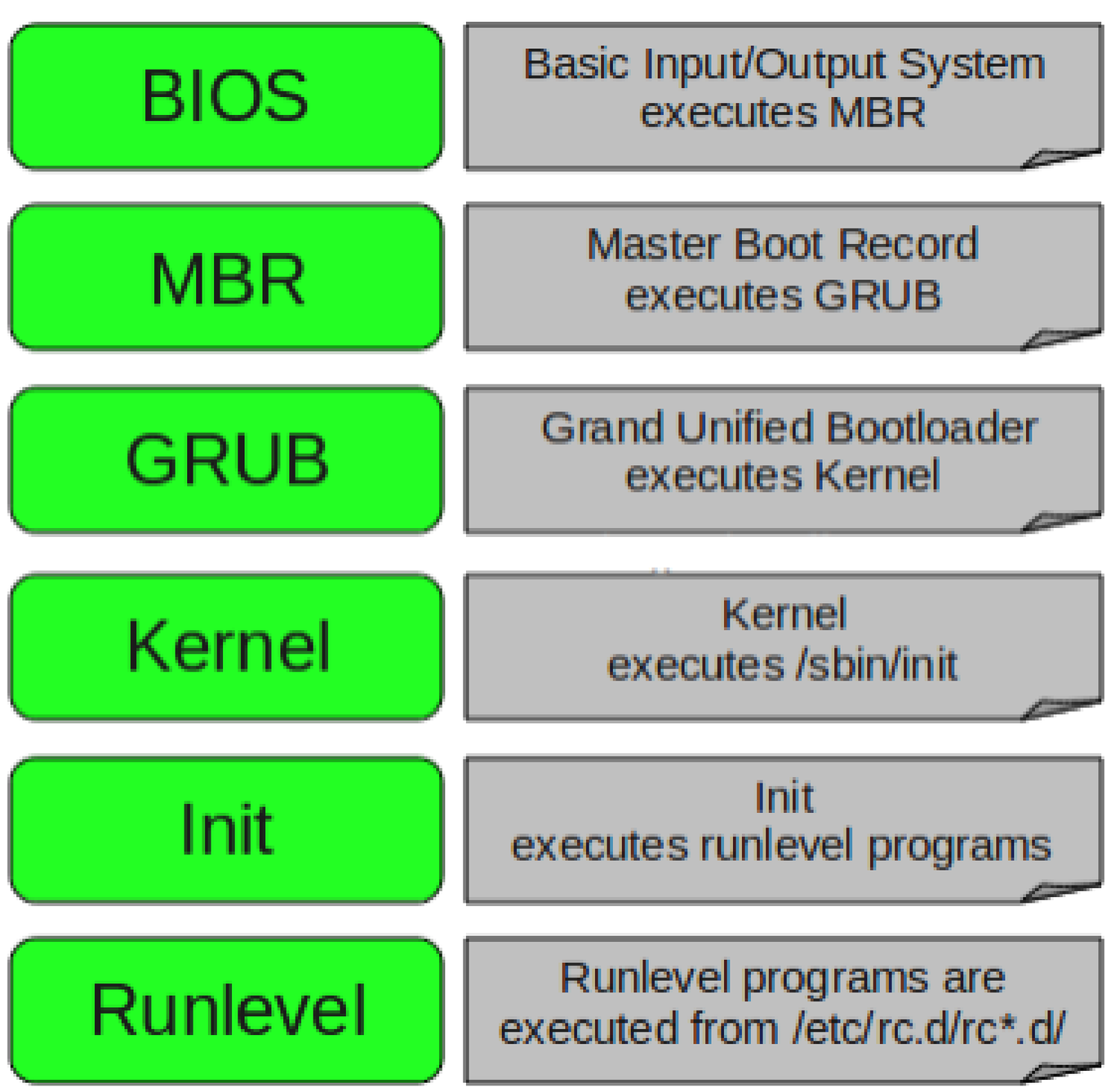


Figure 4.5 Booting Sequence

BIOS

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, CD-ROM, or hard drive. You can press a key during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms, BIOS loads and executes the MBR boot loader.

MBR

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically, /dev/hda, or /dev/sda
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes 3) mbr validation check in last 2 bytes.
- It contains information about GRUB.
- So, in simple terms MBR loads and executes the GRUB boot loader.

GRUB

- GRUB stands for Grand Unified Bootloader.
- GRUB is the typical boot loader for most modern Linux systems.
- If there are multiple kernel images installed on the system, we can choose which one to be executed.
- GRUB displays a splash screen, and waits for a few seconds, if we don‟t enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB knows about the filesystem.
- So, in simple terms GRUB just loads and executes Kernel [18].

Kernel

- The kernel is often referred to as the core of any operating system Linux included.
- It has complete control over everything in your system.
- mounts the root file system that's specified in the grub.conf file.
- it executes the /sbin/init program, which is always the first
- program to be executed. You can confirm this with its process id
- (PID), which should always be 1.
- The kernel then establishes a temporary root file system using Initial RAM Disk (initrd) until the real file system is mounted.

Init

- system executes runlevel programs.
- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels o 0 – halt o 1 – Single user mode o 2 – Multiuser, without NFS o 3 – Full multiuser mode o 4 – unused o 5 – X11 o 6 – reboot.
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate programs.
- Execute „grep initdefault /etc/inittab" on your system to identify the default run level.
- systemd will then begin executing runlevel programs [19].

Runlevel programs

- When the Linux system is booting up, you might see various services getting started. For example, it might say "starting sendmail …. OK". Those are the runlevel programs, executed from the run level directory as defined by your run level
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
- Run level 0 – /etc/rc.d/rc0.d/
- Run level 1 – /etc/rc.d/rc1.d/
- Run level 2 – /etc/rc.d/rc2.d/
- Run level 3 – /etc/rc.d/rc3.d/
- Run level 4 – /etc/rc.d/rc4.d/
- Run level 5 – /etc/rc.d/rc5.d/
- Run level 6 – /etc/rc.d/rc6.d/

Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.

Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.

- Programs that start with S are used during startup. S for startup.
- Programs start with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the Syslog daemon, which has the sequence number 12. S80sendmail is to start the Sendmail daemon, which has the sequence number 80. So, the Syslog program will be started before Sendmail [13].

As explained previously the bootloader in embedded systems is different from a typical laptop, Let's Know about the Raspberry pi boot sequence.

1. When the Raspberry Pi is first turned on, the ARM core is off, and the GPU core is on. At this point the SDRAM is disabled.
2. The GPU starts executing the first stage bootloader, which is stored in ROM on the SoC. The first stage bootloader reads the SD card loads the second stage bootloader (bootcode.bin) into the L2 cache and runs it.
3. bootcode.bin enables SDRAM, reads the third stage bootloader (loader.bin) from the SD card into RAM, and runs it.
4. loader.bin reads the GPU firmware (start.elf).
5. start.elf reads config.txt, cmdline.txt, and kernel.img [20].

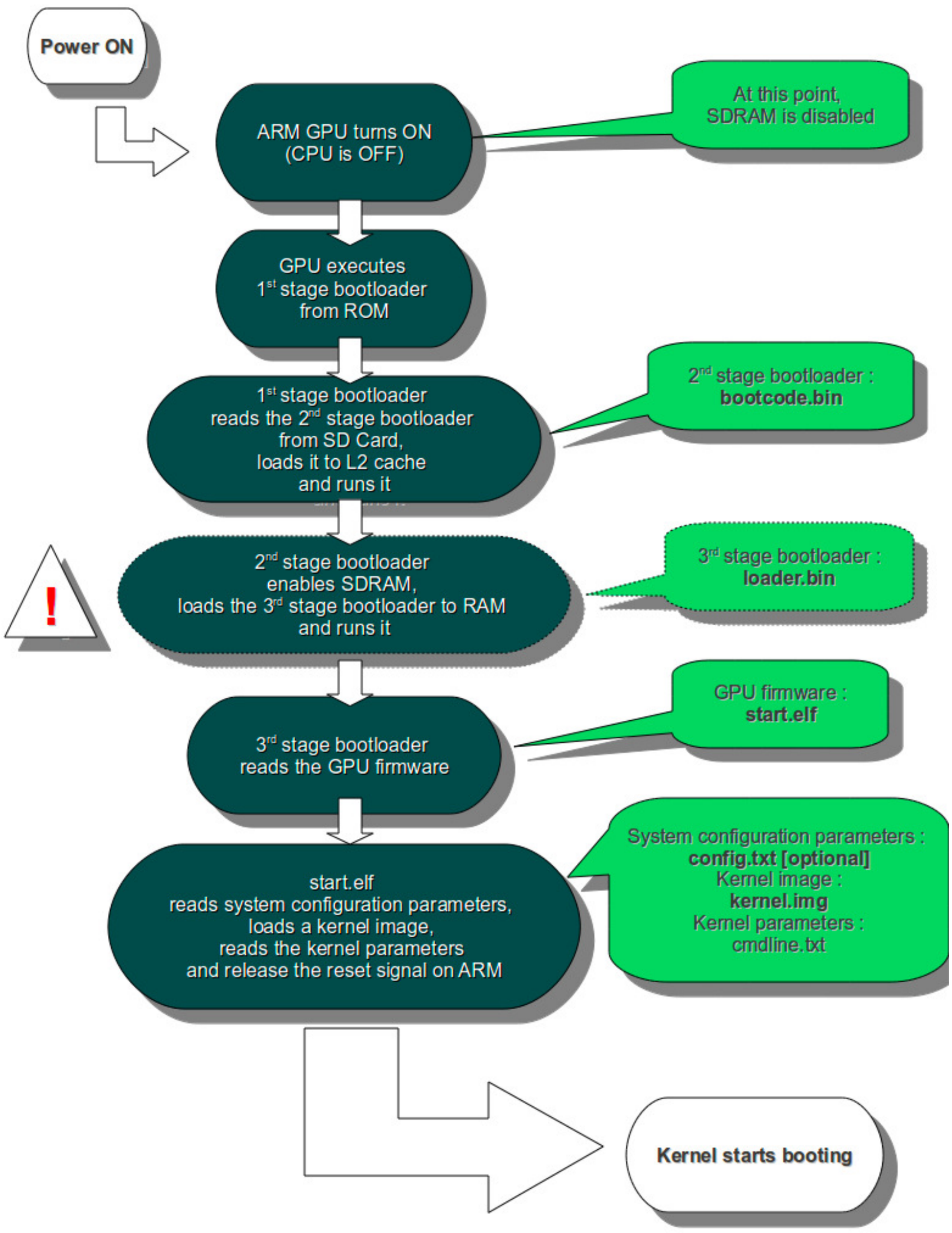


Figure 4.6 Raspberry pi boot sequence

After explaining the booting sequence, let‟s know how to choose a bootloader. Bootloaders come in all shapes and sizes. The kind of characteristics we want from a bootloader is that they are simple and customizable with lots of sample configurations for common development boards and devices. The following table shows a number of them that are in general use:

Name	Architectures
Das U-Boot	ARM, Blackfin, MIPS, PowerPC, SH
Barebox	ARM, Blackfin, MIPS, PowerPC
GRUB 2	X86 , X86 64
RedBoot	ARM, MIPS, PowerPC, SH
CFE	Broadcom MIPS
YAMON	MIPS

Table 4.2 Architectures of different bootloaders

4.4.3 Kernel:

The kernel is the component that is responsible for managing resources and interfacing with hardware, and so affects almost every aspect of our final software build. It is usually tailored to our particular hardware configuration. Once the bootloader loads the Linux kernel into memory and runs it, the Kernel will begin running its startup code. This startup code will initialize the hardware, initialize system critical data structures, initialize the scheduler, initialize all the hardware drivers, initialize filesystem drivers, mount the first filesystem, and launch the first program, among other things. The Linux kernel‟s main job is to start applications and provide coordination among these applications (or programs, as they‟re usually called in Linux). The Linux kernel doesn‟t know about all programs that are supposed to run. So the Linux kernel starts only one program and lets that program launch all the other programs that are needed. This very first program is called the init program, or sometimes just “init” for short. If the kernel can‟t find the init program, the kernel‟s purpose is gone and the kernel crashes. The main difference in the Linux kernel for embedded systems is that it is built to run on a different CPU architecture. Otherwise, the way the kernel operates is consistent with a typical PC, which is one of its strengths [16]

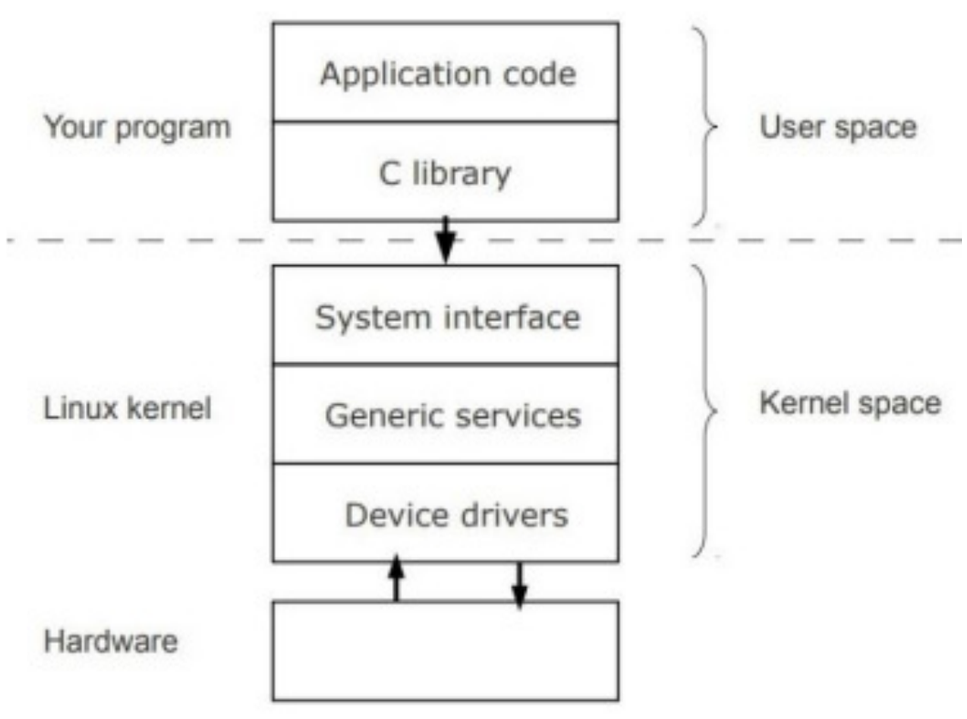


Figure 4.7 Kernel main jobs

Applications running in User space run at a low CPU privilege level. They can do very little other than making library calls.
The primary interface between the User space
and the Kernel space is the C library, which translates user level functions, such as those defined by POSIX, into kernel system calls.

The system call interface uses an architecture ˇ specific method, such as a trap or a software interrupt, to switch the CPU from low privilege user mode to high privilege kernel mode, which allows access to all memory addresses and CPU registers. The System call handler dispatches the call to the appropriate kernel subsystem: memory allocation calls go to the memory manager, filesystem calls to the filesystem code, and so on. Some of those calls require input from the underlying hardware and will be passed down to a device driver. In some cases, the hardware itself invokes a kernel function by raising an interrupt. To choose the kernel for the project, Balancing the desire to always use the latest version of software against the need for vendor-specific additions and an interest in the long-term support of the code base is required [16].

4.4.4 Root Filesystem:

In Linux, the kernel loads programs into memory separately, and the kernel expects these programs to be stored on some medium organized into files and directories. This organization of files and directories is called a filesystem. As is true of many operating systems, Linux has filesystems on media, the data actually stored on a storage medium, and filesystem drivers, the code that knows how to interpret and update the filesystem data on the medium.

In Linux, this medium is often a hard disk. However, embedded systems often don‟t have a hard drive, so the medium can be other hardware devices like SD cards, flash memory, or even RAM to name a few.

Unlike Windows, Linux filesystems get associated with a directory rather than a drive letter. Filesystems can be associated with any directory, even one that is several layers down in a path. This associating a filesystem with a directory is called “mounting.” Linux first starts with an empty directory called / (slash) [13].

During Linux startup, the top most filesystem gets associated with (or mounted to) this directory, and all the contents of that filesystem appear under /. This topmost filesystem is called the root filesystem.

Linux systems expect the root filesystem to be laid out a certain way. So this filesystem is special and can‟t just be some random set of directories and files. This is where directories like bin, sbin, etc, and more come from.
The main point here is that Linux looks for this first program, this init program, to reside in the filesystem.

The root filesystem needs to be created in advance and be mounted to „/" before the kernel can launch the init program. Because embedded systems have different hardware constraints, often Linux embedded systems use special filesystem formats rather than the typical EXT3, EXT4, btrfs, or xfs used on desktop or laptop computers [21].
The first objective is to create a minimal root filesystem that can give us a shell prompt. Then using that as a base we will add scripts to start other programs up, and to configure a network interface and user permissions. Knowing how to build the root filesystem from scratch is a useful skill. To make a minimal root filesystem, we these components:

- init
- Shell
- Daemons
- Shared libraries
- Configuration files
- Device nodes
- Kernel modules

4.4.5 Application:

When the kernel finds, loads and runs the init program, that program then is responsible for bringing up the rest of the system.

The init program is also responsible for starting regular programs. These programs do have user interaction. Embedded systems often have just a few user programs, sometimes just one. In an embedded system, this set of programs makes the device do what it"s supposed to do [20]

4.5 Build Systems:

As explained previously, an embedded Linux system has several major components, all of which are derived from freely available source code and all of which may require varying levels of customization.

we can build individually a toolchain, a bootloader, a kernel, and a root filesystem, and then combine them into a basic embedded Linux system. It has the advantage that we are in complete control of the software, and we can tailor it to do anything we like. If we want it to do something truly odd but innovative, or if we want to reduce the memory footprint to the smallest size possible, building our own is the way to go. But, in the vast majority of situations, building manually is a waste of time and produces inferior, unmaintainable systems. The idea of a build system is to automate all the steps of building the Embedded Linux major components. A capable embedded Linux build system helps us create an embedded Linux distribution tailored to our unique requirements. This must include a cross-toolchain and all the packages required for our project. our build system should be able to generate root file systems in our choice of binary formats, our embedded Linux kernel image with our configuration, a bootloader image, and any other necessary files and utilities so that these can be properly deployed. There are several build systems available, each has its own ideologies and focuses on building the Linux operating system for the target. Below given are the most popular among them:

- **Buildroot**
  This is a build system based on make. This is simple as the GNU build system. Lightweight build system, easy to edit and configure using menu configuration. It supports building several embedded applications with minimal foot-print.
- **Yocto / OpenEmbedded**
  This build system is based on bitbake instead of make. Supports a huge list of packages. Supports several target boards and architectures. Supports a Layered approach to segregate and maintain the build scripts. Became a de-facto build system for embedded Linux. And it is also the base for other Distributions like Peta Linux of Xilinx and RDK for broadband/video (RDK-B). OpenEmbedded has several derivatives which have several features like IDE support, debugging with emulators, etc [13].
- **OpenWRT / LEDE**
  This build system is a popular build system to build custom distributions for networking devices like routers and access points. It is a derivative of buildroot. Has its own lightweight software components as a replacement for the init system(procd), the network manager(netifd), interprocess communication(ubus), etc. it Has seven network-relatedated software components natively supported. It is also the base of a few other distributions like QSDK for Qualcomm network SoCs.

- **AOSP / Soong**
  The Android Open Source Project has built its own build system called Soong which is based on the blueprint (metabuild system) and Kati (Modified Make) and ninja (alternative for make). Soong builds an Android system with lots of Android-specific components like its own init system, HAL codes, system services, Java packages, and external opensource packages.

4.6 The Yocto Project:

The Yocto Project is more general in the way it defines the target system, so it can build complex embedded devices. Every component is generated as a binary package, by default, using the RPM format, and then the packages are combined together to make the filesystem image. Furthermore, you can install a package manager in the filesystem image, which allows you to update packages at runtime. In other words, when you build with the Yocto Project, you are, in effect, creating your own custom Linux distribution.

The Yocto Project is primarily a group of recipes, similar to Buildroot packages but written using a combination of Python and shell script, together with a task scheduler called BitBake that produces whatever you have configured, from the recipes. The Yocto Project is structured with cross-platform tools, and metadata, to enable developers rapidly to create customized Linux distributions from source code, which simplifies the development process. Compared to a full Linux distribution, the customized system will reserve the software you need to make the system much more specific to your application [22].

Yocto Project has advantages in system and application development, archiving, and management. Developers can customize their systems in terms of speed, memory footprint, and even memory utilization. Yocto Project allows software customization and construction exchange for multiple hardware platforms, and also maintains a software stack in scale.

The Yocto Project collects together several components, the most important of which are the following:

- OE-Core: is metadata composed of basic recipes, classes, and related files. These metadata are designed to be common in many different OpenEmbedded derived systems (including Yocto projects).
- BitBake: The core tool of the OpenEmbedded build system. BitBake plays the role of build a system engine and is responsible for parsing metadata, generating task lists from it, and then performing these tasks.

- Poky: a reference distribution. Poky is the name of the reference distribution or reference OS of the Yocto Project. Poky includes the OpenEmbedded Build System (Bitbake and OpenEmbedded-Core) and a set of metadata to help us start building our own distribution. Poky uses OpenEmbedded Build System to build a small embedded operating system. Poky is an integration layer on top of OE-Core. Poky provides the following:
    - A basic level of distro infrastructure to illustrate how to customize the distro.
    - A means to verify the Yocto Project components.
- Documentation: This is the user's manuals and developer's guides for each component.
- Toaster: This is a web-based interface to Bitbake and its metadata.
- ADT Eclipse: This is a plugin for Eclipse.

The Yocto Project provides a stable base, which can be used as it is or can be extended using meta layers. When downloading the build system, the Poky build „file" is called a recipe and a layer. we can modify, assign, or anyway we need to create our own customized embedded Linux [16].

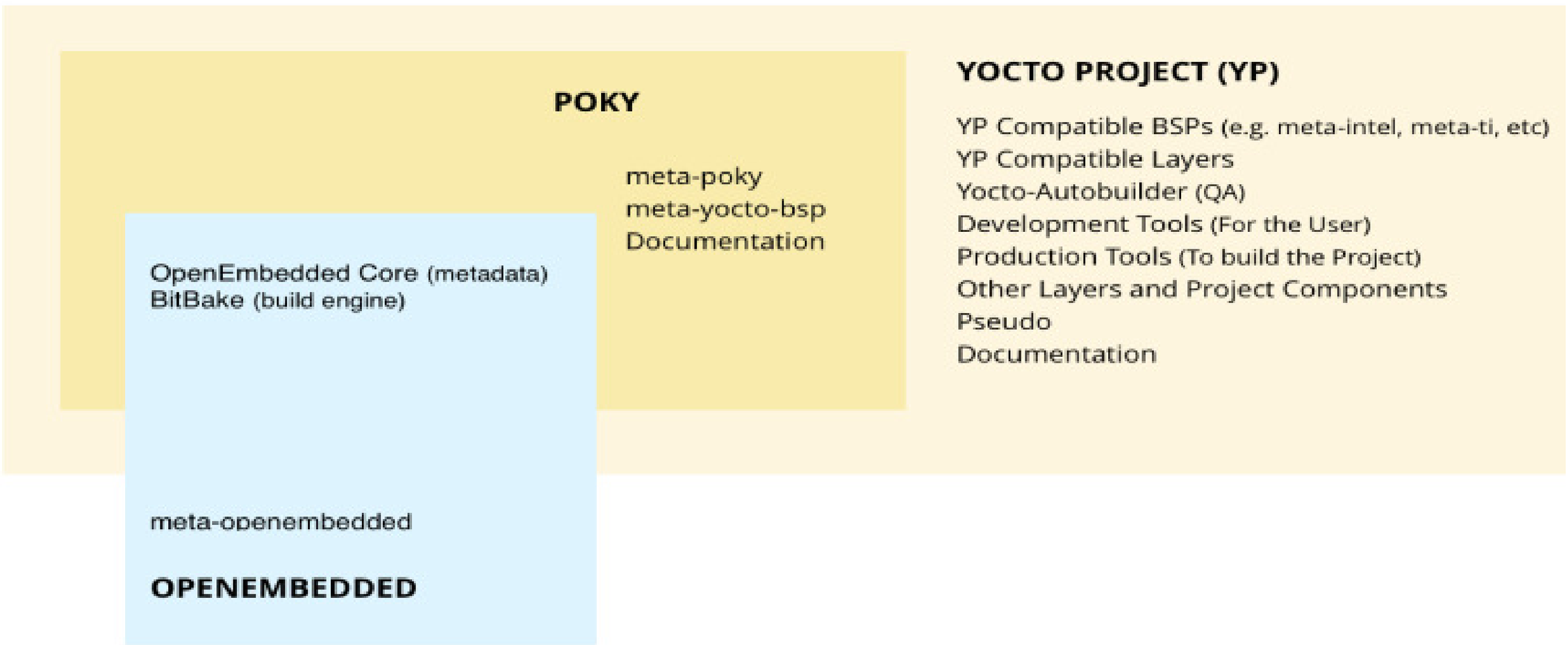


Figure 4.9 Yocto project components

Some Yocto Project Terminology:

Recipe: The most common form of metadata. Recipe contains a list of settings and tasks used to build a binary image file. Recipe describes where you get the code and which (code) patch you need to apply. At the same time, Recipe also describes the dependencies on other recipes or libraries, as well as configuration and compilation options. Recipe is stored in Layer.

- **Layer:** A collection of related recipes. A Layer is a repository containing relevant metadata that tells the OpenEmbedded build system how to build the target. Yocto Project‟s layer model promotes collaboration, sharing, customization, and reuse in the Yocto Project development environment. Layers logically separate the information of your project.
- **Metadata:** Layer contains the recipe files, patches, and additional files provided by the user, other information referring to the build instructions, and data that controls what and how to build. A good example of the software layer might be the meta-Qt5 Layer from the OpenEmbedded Layer Index.
- **A Board Support Package (BSP):** is a collection of information that defines how to support a particular hardware device, set of devices, or hardware platform. The BSP includes information about the hardware features present on the device and kernel configuration information along with any additional hardware drivers required.
- **Configuration Files:** Files that hold global definitions of user-defined variables and hardware configuration information. They tell the build system what to build and put into the image to support a particular platform.
- **Packages:** The output of the build system used to create your final image [22].

OpenCV Subsystem

5

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library that provides a wide range of tools and algorithms for analyzing, processing, and manipulating images and videos. It was initially developed by Intel in 1999 and is now maintained by OpenCV.org.

The library is written in C & C++ and has bindings for several programming languages, including Python, Java, and MATLAB. OpenCV supports various platforms, including Windows, Linux, macOS, Android, and iOS.

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.[24]]

5.1 Who uses openCV?

OpenCV is used by a wide range of individuals and organizations, including researchers, developers, hobbyists, and companies.

Researchers: OpenCV is widely used in the field of computer vision research. It provides a powerful and flexible platform for developing and testing new algorithms and techniques.

Developers: OpenCV is used by developers in various industries, including robotics, automation, and autonomous vehicles. It provides tools and algorithms for image and video processing, which are essential for many applications.

Hobbyists: OpenCV is popular among hobbyists who are interested in computer vision and machine learning. It provides an accessible platform for learning and experimenting with these technologies.

Companies: OpenCV is used by many companies in various industries, including healthcare, entertainment, and security. It provides a powerful and flexible platform for developing and deploying computer vision applications.

5.2 Computer Vision:

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information.

5.2.1 What is computer vision:

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware. Computer Vision overlaps significantly with the following fields:

- Image Processing: It focuses on image manipulation.
- Pattern Recognition: It explains various techniques to classify patterns.
- Photogrammetry: It is concerned with obtaining accurate measurements from images.[25]

5.2.2 What is image processing:

Image processing is the process of analyzing and manipulating digital images to improve their quality or extract useful information from them. It involves applying mathematical algorithms and techniques to digital images in order to enhance, transform, or extract features from the images.

Image processing can be divided into two main categories: **analog and digital**.

- **Analog image processing** involves using analog devices, such as filters and lenses, to manipulate images.
- **Digital image processing**, on the other hand, involves using computer algorithms to manipulate digital images.

Some common tasks in digital image processing include:

- **Image enhancement**: This involves improving the visual quality of an image by adjusting its brightness, contrast, color balance, and other parameters.
- **Image restoration:** This involves removing noise, blurring, or other defects from an image to restore its original quality.
- **Image segmentation:** This involves dividing an image into multiple regions or segments based on their characteristics, such as color or texture.
- **Object detection and recognition:** This involves identifying and locating objects within an image, and recognizing them based on their shape, color, or other features.
- **Feature extraction:** This involves identifying and extracting specific features from an image, such as edges, corners, or other patterns.

5.2.3 Computer vision vs Image processing:

- **Image processing** deals with image-to-image transformation. The input and output of image processing are both images.
- **Computer vision** is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.[25]

5.3 Applications of computer vision:

Computer Vision gives the machines the sense of sight—it allows them to "see" and explore the world thanks to Machine Learning and Deep Learning algorithms.
This powerful technology has quickly found applications across multiple industries, becoming an indispensable part of technological development and digital transformation.[26]

5.3.1 Transportation:

The increasing demands of the transportation sector have propelled technological development in this industry, with computer vision at its center. From autonomous vehicles to parking occupancy detection, the Intelligent Transportation System (ITS) has become a critical field for promoting the efficiency, effectiveness, and safety of transportation.[26]

5.3.1.a Self-driving cars:

In 2023 autonomous vehicles are no longer science fiction. In fact— Thousands of engineers and developers worldwide are already testing and improving the reliability and safety of self-driving cars. Computer vision is used to detect and classify objects (e.g., road signs or traffic lights), create 3D maps or motion estimation, and played a key role in making autonomous vehicles a reality.

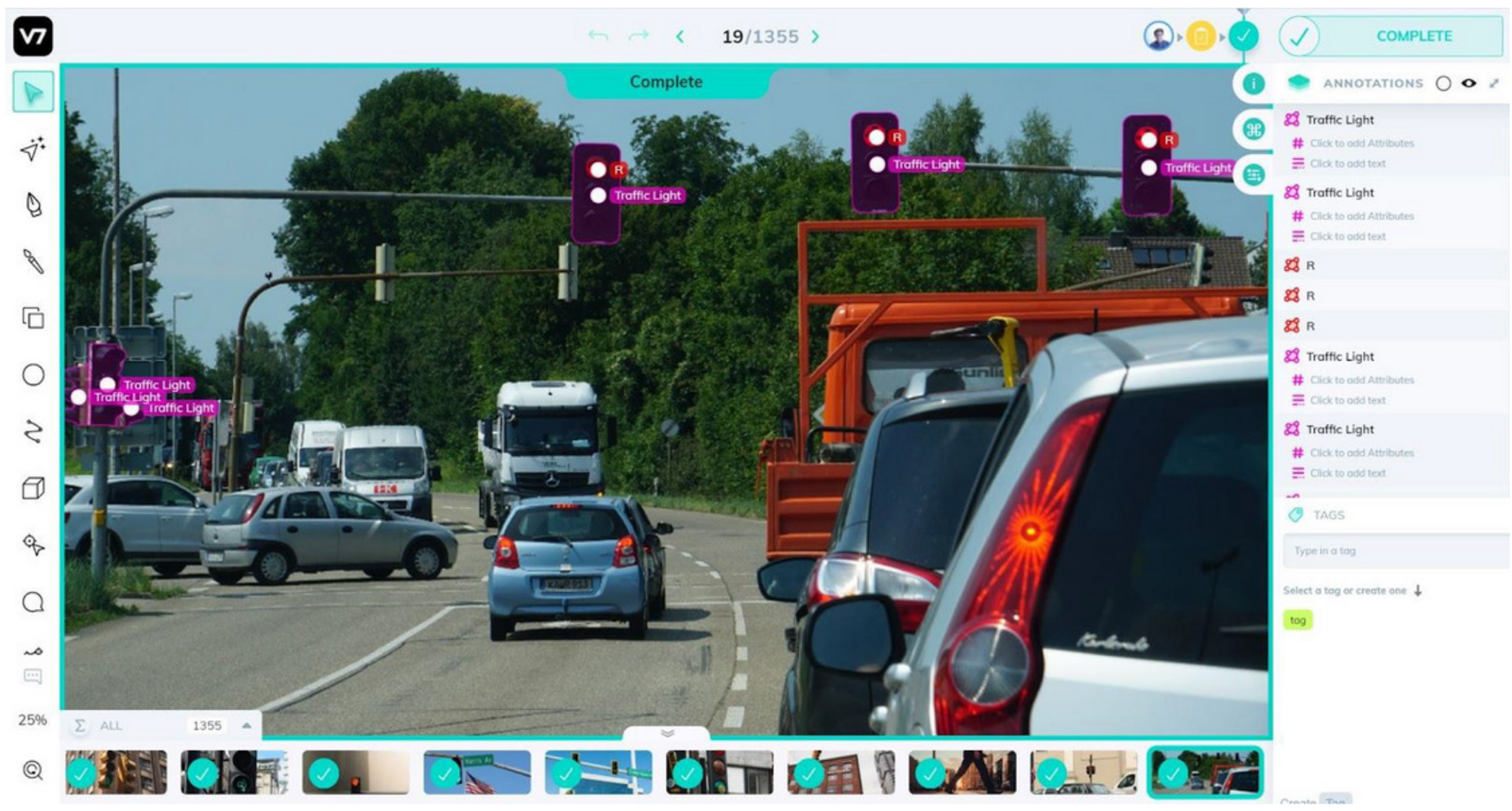


Figure 5.1 self-driving cars

Self-driving cars collect data on their surroundings from sensors and cameras, interpret it, and respond accordingly. Researches working on the ADAS technology combine computer vision techniques such as pattern recognition, feature extraction, object tracking, and 3D vision to develop real-time algorithms that assist driving activity. Here's how AI traffic light detection works in real life.[26]

5.3.1.b Pedestrian detection:

Pedestrian detection is a very important area of research because it can enhance the functionality of a pedestrian protection system in Self Driving Cars. We can extract features like head, two arms, two legs, etc, from an image of a human body and pass them to train a machine learning model. After training, the model can be used to detect and track humans in images and video streams. However, OpenCV has a built-in method to detect pedestrians. It has a pre-trained HOG(Histogram of Oriented Gradients) + Linear SVM model to detect pedestrians in images and video streams.[27]

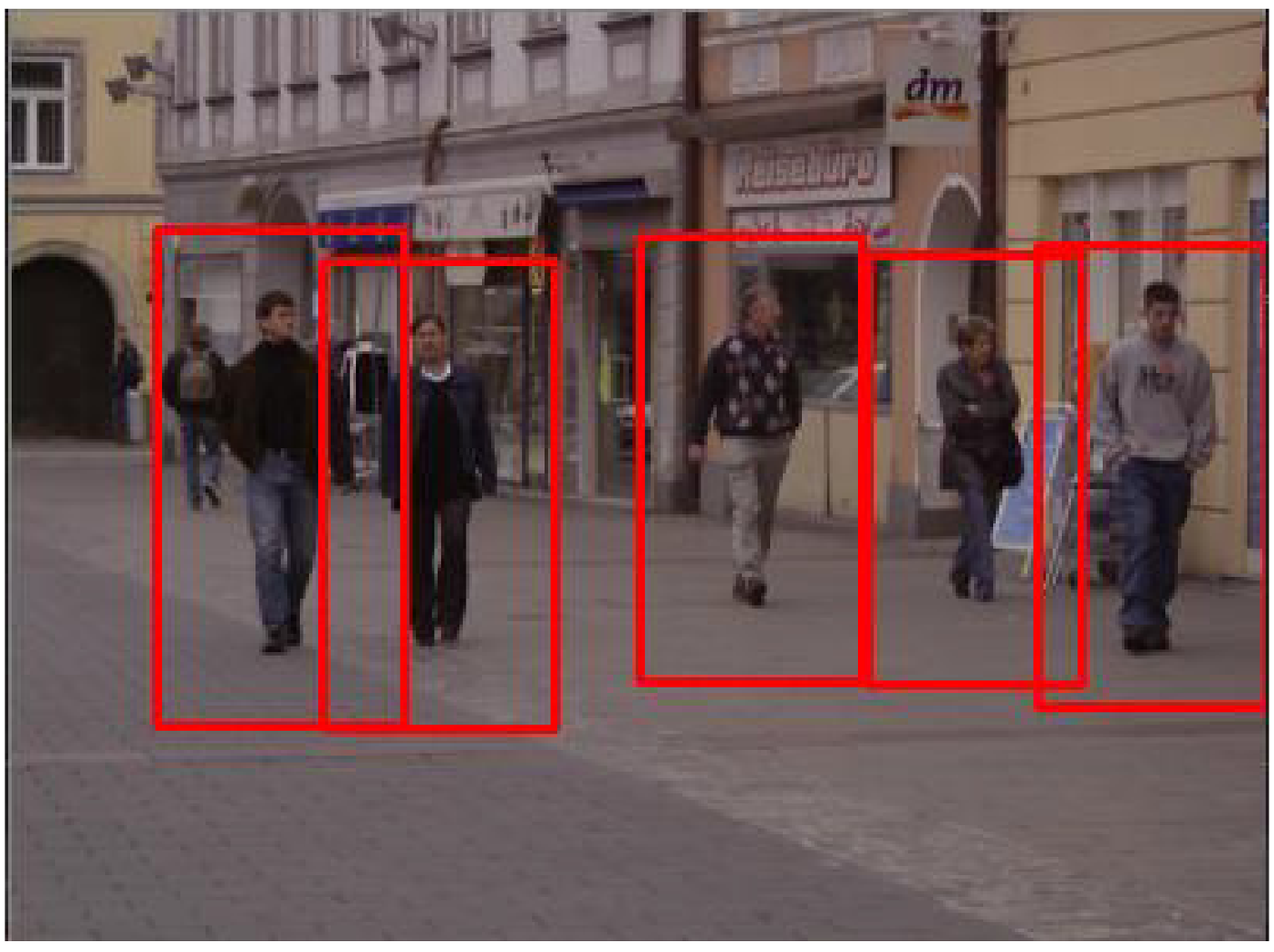


Figure 5.2 pedestrian detection

5.3.1.c traffic flow analysis:

Drone and camera-based traffic flow tracking and estimation have also been made possible thanks to the developments in the field of computer vision. The algorithms can now accurately track and count freeway traffic or monitor and analyze traffic density in urban areas (e.g., at intersections), helping to design better traffic management systems and improve road safety.[26]

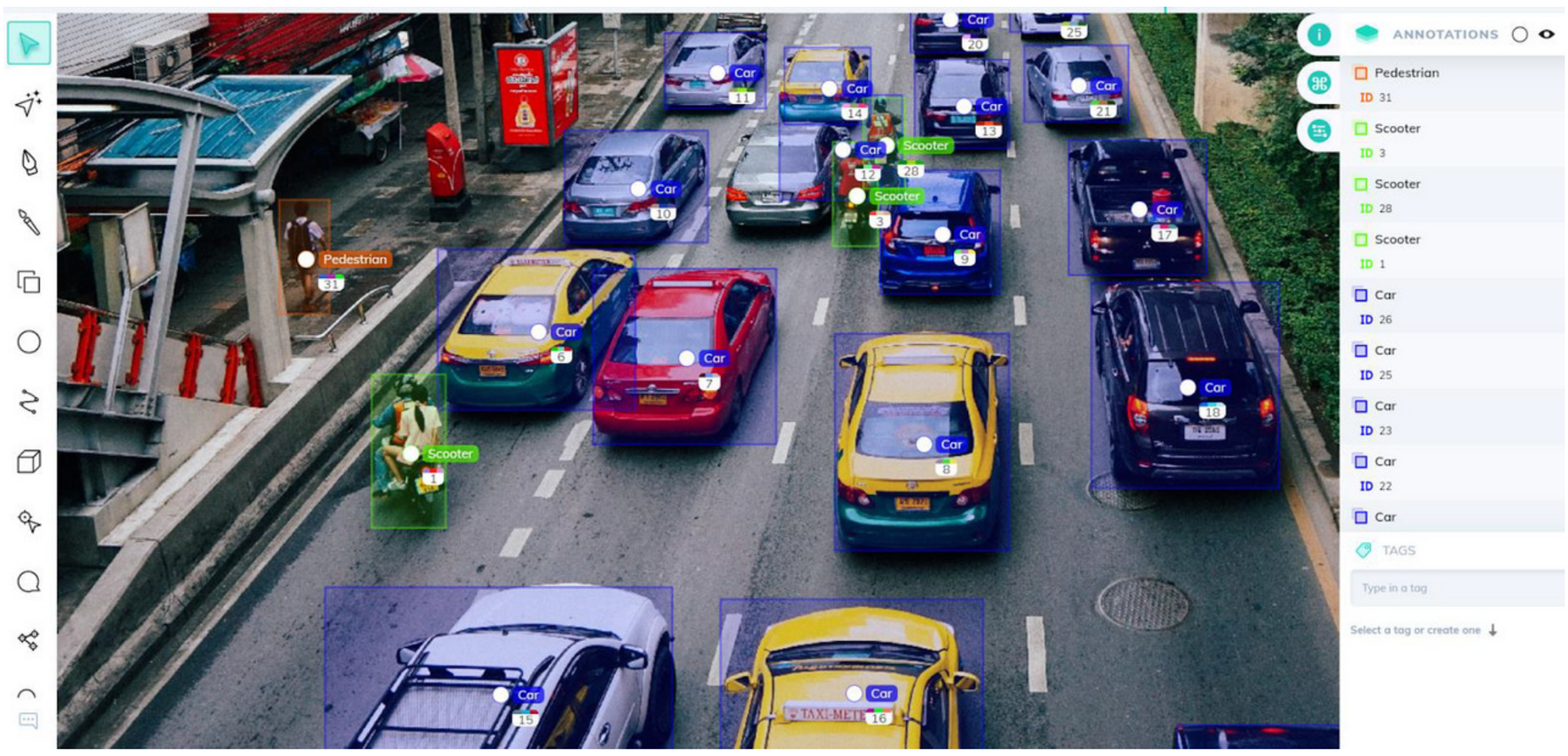


Figure 5.3 traffic flow analysis

5.3.2 Traffic flow analysis medical imaging:

OpenCV is used in medical imaging applications to analyze and process medical images, such as X-rays, CT scans, and MRIs. It provides tools and algorithms for image segmentation, feature extraction, and object detection, which are essential for medical diagnosis and treatment.

Telem**edicine**: OpenCV is used in telemedicine 6applications to enable remote diagnosis and treatment of patients. It provides tools for real-time video processing and analysis, which can be used to monitor patients and provide medical advice remotely.

Surgical robotics: OpenCV is used in surgical robotics to enable precise and accurate surgical procedures. It provides tools for image-guided surgery, which can help surgeons navigate and manipulate surgical instruments with greater precision.

Assistive technologies: OpenCV is used in assistive technologies for people with disabilities, such as visual aids for the blind. It provides tools for image recognition and object detection, which can be used to help people navigate their environment more easily.

Healthcare analytics: OpenCV is used in healthcare analytics to analyze large amounts of medical data, such as electronic health records and medical images. It provides tools for data mining, pattern recognition, and machine learning, which can be used to identify trends and patterns in medical data.

5.4 OpenCV Library:

openCV is a massive open-source library for various fields like computer vision, machine learning, image processing and plays a critical function in real-time operations, which are fundamental in today's systems.

5.4.1 Features of OpenCV library:

One of the key features of OpenCV is its ability to provide access to the source code, which means that developers can modify and customize the library to suit their specific needs. This feature is particularly useful for researchers and developers who want to explore and experiment with cutting-edge computer vision algorithms and techniques.

OpenCV provides its source code through a GitHub repository, which can be accessed at https://github.com/opencv/opencv. This repository contains the complete source code for the library, along with documentation, examples, and other resources.

In addition to the main OpenCV repository, there are also several other related repositories that provide additional functionality and tools. These include opencv_contrib, which contains additional modules and algorithms contributed by the community, and opencv_extra, which provides additional datasets and examples.

Using OpenCV library, you can:

- Read and write images
- Capture and save videos
- Process images (filter, transform)
- Preform feature detection
- Detect specific objects such as faces, eyes, cars, in the videos or images
- Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it

OpenCV was originally developed in C++. In addition to it, Python and Java bindings were provided. OpenCV runs on various Operating Systems such as windows, Linux, OSx, FreeBSD, Net BSD, Open BSD, etc.

5.5 What is needed for installation:

To install OpenCV, you will need the following:

1. C++ compiler: OpenCV is written in C++, so you will need a C++ compiler to build the OpenCV libraries. Some popular C++ compilers include GCC, Clang, and Microsoft Visual Studio.
2. CMake: CMake is a build system generator that is used to build OpenCV. You will need to install CMake on your system in order to build OpenCV.
3. Image and video codecs: OpenCV relies on external image and video codecs to work with various image and video formats. You will need to install the necessary image and video codecs on your system.
4. Libraries: OpenCV uses a number of external libraries, such as NumPy, Matplotlib, and Qt, to provide additional functionality. You will need to install these libraries on your system in order to use them with OpenCV.
5. Operating system: OpenCV is a cross-platform library that can be installed on various operating systems, including Windows, Linux, and macOS. You will need to choose the appropriate installation method for your operating system.

5.6 Python interface for OpenCV:

OpenCV provides a Python interface that allows you to use OpenCV functions and algorithms in your Python programs. Here are some of the methods provided by the OpenCV Python library:

imread(): This method is used to read an image from a file and store it as a NumPy array.

imshow(): This method is used to display an image on-screen. It takes a window name and a NumPy array as input.

imwrite (): This method is used to write an image to a file. It takes a filename and a NumPy array as input.

resize (): This method is used to resize an image. It takes a NumPy array and the desired size as input.

ctvColor(): This method is used to convert an image from one color space to another. It takes a NumPy array and the desired color space conversion as input.

threshold(): This method is used to threshold an image. It takes a NumPy array and the desired threshold value as input.

Canny(): This method is used to detect edges in an image using the Canny edge detection algorithm. It takes a NumPy array and the desired threshold values as input.

findContours (): This method is used to find contours in an image. It takes a NumPy array and the desired contour retrieval mode and contour approximation method as input.

drawContours (): This method is used to draw contours on an image. It takes a NumPy array, a list of contours, and the desired contour index as input.

HoughLines (): This method is used to detect lines in an image using the Hough transform. It takes a NumPy array and the desired Hough transform parameters as input.

5.7 Handling Files, Cameras, and GUIs:

Installing OpenCV and running samples is fun, but at this stage, we want to try things out in our own way. This chapter introduces OpenCV's I/O functionality. We also discuss the concept of a project and the beginnings of an object-oriented design for this project, which we will flesh out in subsequent chapters.

By starting with a look at I/O capabilities and design patterns, we will build our project in the same way we would make a sandwich: from the outside in. Bread slices and spread, or endpoints and glue, come before fillings or algorithms. We choose this approach because computer vision is mostly extroverted—it contemplates the real world outside our computer—and we want to apply all of our subsequent algorithmic work to the real world through a common interface.

5.7.1 Reading/writing an image file:

OpenCV provides the imread() and imwrite() functions that support various file formats for still images. The supported formats vary by system but should always include the BMP format. Typically, PNG, JPEG, and TIFF should be among the supported formats too.[28]

5.7.2 Reading/writing on a video:

OpenCV provides a Video I/O module that allows you to read and write video files. The Video I/O module supports various video formats, codecs, and containers, and provides a simple and convenient interface for working with video files.

Here are some of the methods provided by the Video I/O module for reading and writing videos in OpenCV:

VideoCapture (): This method is used to open a video file for reading. It takes the filename as input and returns a VideoCapture object that can be used to read frames from the video.

isOpened (): This method is used to check if a VideoCapture object is open. It returns True if the object is open and False otherwise.

read (): This method is used to read a frame from a VideoCapture object. It returns a tuple containing the frame and a boolean value indicating whether the read was successful.

videoWriter (): This method is used to create a VideoWriter object for writing a video file. It takes the filename, codec, frame rate, and frame size as input.

write (): This method is used to write a frame to a VideoWriter object. It takes the frame as input and writes it to the video file.

release (): This method is used to release a VideoCapture or VideoWriter object. It closes the associated file and frees any resources used by the object.

To read frames from a video file, you can use the **videoCapture()** method to open the file, and then use the **read()** method to read frames one by one. To write frames to a new video file, you can use the **VideoWriter()** method to create a new file, and then use the **write()** method to write frames to the file.
Finally, you can use the **release()** method to close the files and release any resources used by the VideoCapture and VideoWriter objects.

Example:

```
import cv2
# Open the input video file
cap = cv2.VideoCapture('input_video.mp4')

# Get the video frame rate and size
frame_rate = int(cap.get(cv2.CAP_PROP_FPS))
frame_size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))

# Create a VideoWriter object for the output
out = cv2.VideoWriter('output_video.avi', cv2.VideoWriter_fourcc(*'XVID'),
frame_rate, frame_size)

# Loop through the frames in the input video
while cap.isOpened():
    # Read a frame from the input video
    ret, frame = cap.read()
    # If the read was successful, process the frame
    if ret:
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Write the processed frame to the output video
    out.write(gray)
    # Display the processed frame on-screen
    cv2.imshow('Processed Frame', gray)

# If the read was unsuccessful, break out of the loop
else:
    break
# Wait for a key press to exit
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
# Release the VideoCapture and VideoWriter objects
cap.release()
out.release()
# Close all windows
cv2.destroyAllWindows()
```

In this example, we first open the input video file using the **videoCapture()** method and get the frame rate and size of the video. We then create a VideoWriter object for the output video file using the Video**Writer()** method.

Next, we loop through the frames in the input video using a while loop. For each frame, we read it from the input video using the **read()** method, convert it to grayscale using the **cvtColor()** method, and write the processed frame to the output video using the **write()** method. We also display the processed frame on-screen using the **imshow()** method.

Finally, we release the VideoCapture and VideoWriter objects using the **release()** method and close all windows using the **destroyAllWindows()** method.

5.7.3 Camera with openCV:

OpenCV provides a simple and convenient interface for working with cameras in Python. Here are the basic steps for accessing a camera using OpenCV:

Create a VideoCapture object: To access a camera, you first need to create a VideoCapture object. You can do this by passing the camera index (usually 0 for the default camera) as an argument to the VideoCapture() method.

Check if the camera is open: After creating the VideoCapture object, you should check if the camera is open using the isOpened() method. If the camera is not open, you may need to check if another application is using the camera or if the camera is connected properly.

Read frames from the camera: Once the camera is open, you can use the read() method to read frames from the camera. This method returns a boolean value indicating whether the read was successful, and a frame (as a NumPy array) representing the current image from the camera.

Release the camera: When you are finished using the camera, you should release it using the release() method. This frees any resources used by the VideoCapture object and closes the camera.

Example:

```
import cv2
# Create a VideoCapture object for the default camera (index 0)
cap = cv2.VideoCapture(0)

# Check if the camera is open
if not cap.isOpened():
        print("Could not open camera")
        exit()
# Loop through the frames from the camera
while True:
        # Read a frame from the camera
        ret, frame = cap.read()
        # If the read was unsuccessful, break out of the loop
        # If the read was unsuccessful, break out of the loop
                break

        # Display the frame on-screen
        cv2.imshow('Camera', frame)
        # Wait for a key press to exit
        if cv2.waitKey(1) & 0xFF == ord('q'):
                break

# Release the camera and close the window
cap.release()
cv2.destroyAllWindows()
```

This code creates a VideoCapture object for the default camera (index 0), checks if the camera is open, and then loops through the frames from the camera. For each frame, it reads the frame using the read() method, displays the frame on-screen using the imshow() method, and waits for a key press to exit. Finally, it releases the camera using the release() method and closes the window using the destroyAllWindows() method.

5.8 Haar cascade and YOLO:

- Haar Cascade and YOLO (You Only Look Once) are two popular object detection algorithms that are used to identify objects in images or videos.

- Haar Cascade is a machine learning-based object detection algorithm that was proposed by Viola and Jones in 2001. It uses Haar wavelets to extract features from images and then applies a cascade of classifiers to detect objects. Haar Cascade is a relatively simple and efficient algorithm that can run in real-time on low-power devices. It is commonly used for face detection in images and videos, but it can also be trained to detect other objects such as cars, pedestrians, and animals.

- On the other hand, YOLO is a deep learning-based object detection algorithm that was proposed by Redmon et al. in 2015. It uses a single neural network to predict the bounding boxes and class probabilities of objects in an image. YOLO is faster than other deep learning-based object detection algorithms such as Faster R-CNN and SSD, and can run in real-time on high-end GPUs. It is commonly used for detecting objects in real-time video streams and is known for its accuracy and speed.

- The main difference between Haar Cascade and YOLO is that Haar Cascade is a machine learning-based algorithm, while YOLO is a deep learning-based algorithm. Haar Cascade is relatively simple and efficient, but may not be as accurate as YOLO for complex object detection tasks. On the other hand, YOLO is more accurate and can handle complex object detection tasks, but requires more computational resources and is more complex to implement.

- Both Haar Cascade and YOLO are widely used in computer vision applications and have their own strengths and weaknesses. The choice between them depends on the specific requirements of the application and the available resources.

- OpenCV supports both Haar Cascade and YOLO object detection algorithms, allowing you to use these algorithms in your computer vision applications.

C

TRAFFIC LIGHTS SYSTEM

Chapter 6: Traffic lights with Ambulance system

Traffic lights with Ambulance system

6

Traffic congestion is one of the many problems that the world faces due to increased population and rapid growth in the number of vehicles. In countries like India, where the rate of road expansion is only one-third of the vehicular growth rate, traffic congestion has become a severe problem. The current annual growth of vehicles is around 11%, while the annual road extension remains around 4%. This has several adverse effects, including delayed services, wastage of fuel, and damage to the environment. Studies show that traffic congestion causes the wastage of 2.5 lakhs of liters of non-renewable fuel in one day [30].

Our project aims to address the severe impact of traffic congestion on the emergency vehicle transportation system, particularly in countries like India and Thailand where road width and length make it impossible to create separate lanes for emergency vehicles. Ambulances face difficulties navigating through traffic, which can have critical consequences for patients.

Existing ideas for addressing traffic congestion include controlling traffic lights using timers or electronic sensors to detect vehicles and produce signals that cycle. Fuzzy logic networks are also used, but these systems require the presence of traffic policemen during peak traffic hours and do not cater to the necessities of emergency vehicles like ambulances, where every second counts [31].

The main objectives of our project are as follows:

- To adjust the timing of traffic signals based on a traffic threshold value.
- Detecting emergency vehicles such as fire trucks, police cars, and ambulances, and working to clear the road for them.

In our project we used openCV to detect the ambulance by using Haar Cascade. A camera is placed on the traffic light which rotates in all directions in order to detect ambulances. The system which is on the traffic light contains Raspberry pi , stepper motor and a camera , in which the Raspberry pi controls the stepper , with the camera attached to it , to rotate in every direction. The location of the ambulance is sent to the Raspberry pi using UART communication protocol.

6.1 Methodology:

- Ambulance Detection
- Decisions Based on Data

6.2 Number of the ways that can be used to determine the traffic signal for an ambulance?:

These are some of the ways that can be used to determine the traffic signal for an ambulance. Each application can have unique features and different challenges, and careful study of city requirements and available resources is needed when implementing these systems [32].

Each method of determining the traffic signal for an ambulance comes with its own set of challenges, including:

- Using sensors Like (Sound Sensor, camera)
- Using wireless communication technology
- Using Sensor and communication technology
- Using smart internet and cloud technology
- Using smartphone applications

6.2.1 Using Camera:

One of the challenges of using sensors is ensuring that the sensors are accurate and reliable. False positives or false negatives can lead to delays or accidents. Another challenge is ensuring that the sensors are installed in the right locations to detect the ambulance at the right time.

Used in this way:

- camera with Raspberry pi
- Arduino or Stm32 or AVR to control the traffic light

Steps:

- The traffic signal will operate as usual.
- A camera will be placed on the traffic signal at a suitable location and will rotate on the roads to detect the location of the ambulance and send the data to the traffic signal control unit.
- Then, the traffic signal control unit will start to clear the road for the ambulance.

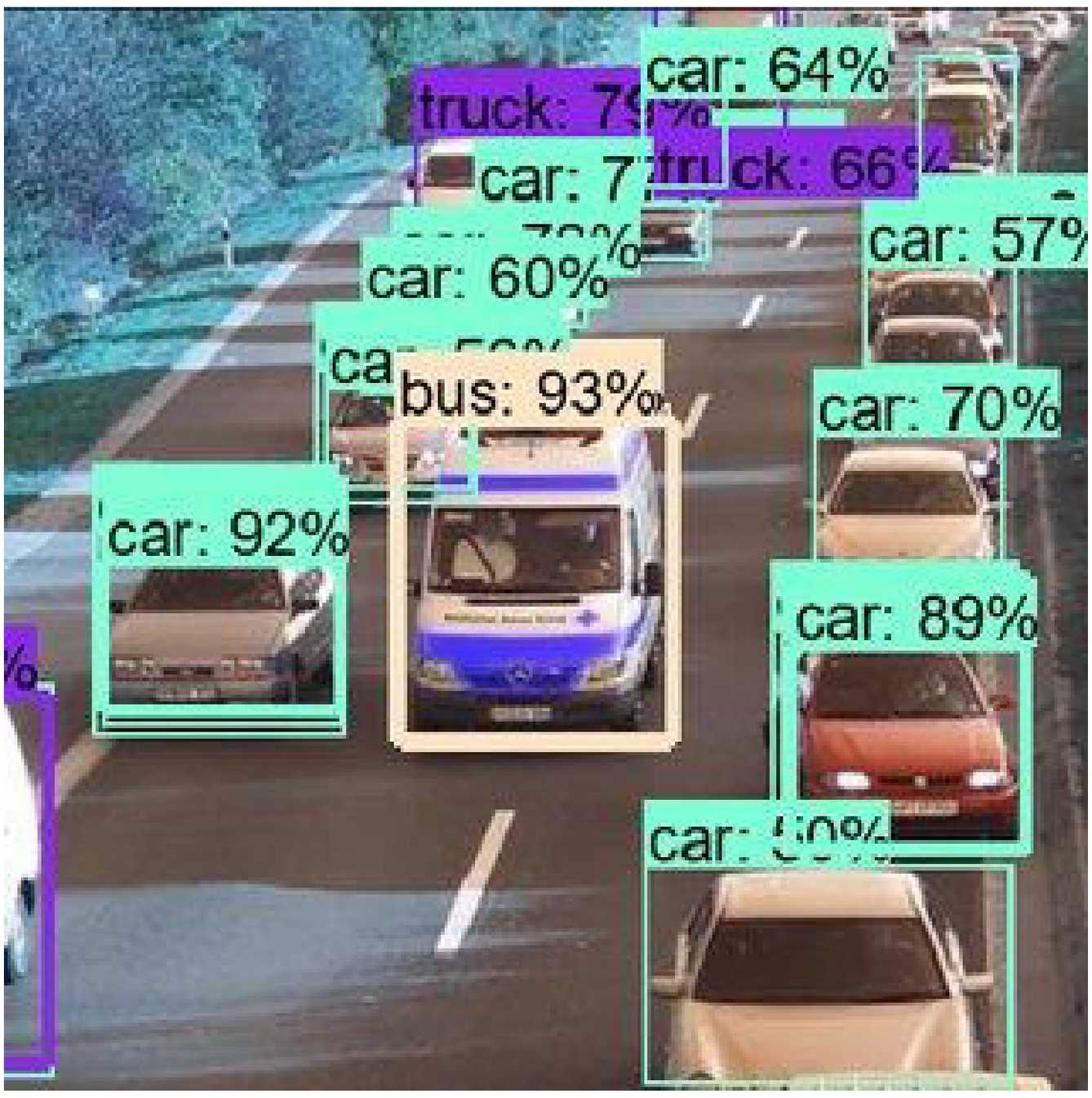


Figure 6.1 detect cars using camera

Advantage:

Using a camera in conjunction with a traffic light system has several advantages, including:

1. Improved accuracy: A camera can provide more accurate and real-time data on the location of emergency vehicles, allowing traffic signals to respond more quickly and accurately to changing traffic conditions.
2. Increased safety: By detecting emergency vehicles in real time, traffic signals can improve safety by reducing the chance of accidents or collisions between emergency vehicles and other vehicles on the road.
3. Better traffic flow: By responding quickly and accurately to emergency vehicles, traffic signals can help to minimize disruption to traffic flow and reduce congestion on the road.
4. Cost-effective: Using a camera-based system can be more cost-effective than other methods, such as installing sensors in the road or equipping emergency vehicles with expensive communication devices.
5. Flexibility: A camera-based system is more flexible and can be easily installed in different locations, making it suitable for various traffic conditions and road layouts.

Overall, using a camera in conjunction with a traffic light system can provide more accurate, efficient, and cost-effective traffic management in emergency situations, while also offering greater flexibility in implementation.

Disadvantage:

Using a camera with a traffic light system to detect an ambulance also has some potential disadvantages, including:

1. Limited range: Cameras have limited range, and may not be able to detect an ambulance far enough away to allow the traffic signal to respond in a timely manner.
2. Dependence on visibility: Cameras rely on visibility, and may not be able to detect an ambulance in adverse weather conditions or at night.
3. Privacy concerns: The use of cameras in public areas may raise privacy concerns among some individuals.
4. Maintenance and repair: Cameras require regular maintenance and repair to ensure they are functioning properly, and may be subject to damage or vandalism.
5. Cost: While using a camera-based system can be cost-effective in some cases, the initial installation costs of a camera system may be higher than other methods of detecting an ambulance.

Overall, while using a camera with a traffic light system to detect an ambulance has several advantages, it also has some potential disadvantages that should be considered when implementing such a system.

6.2.2 Using wireless communication technology:

The main challenge of using wireless communication technology is ensuring that the communication is reliable and secure. Interference or signal loss can lead to delays or miscommunication, which can be dangerous in emergency situations. Another challenge is ensuring that the wireless device is compatible with the traffic signal controller and that the necessary infrastructure is in place to support the communication.

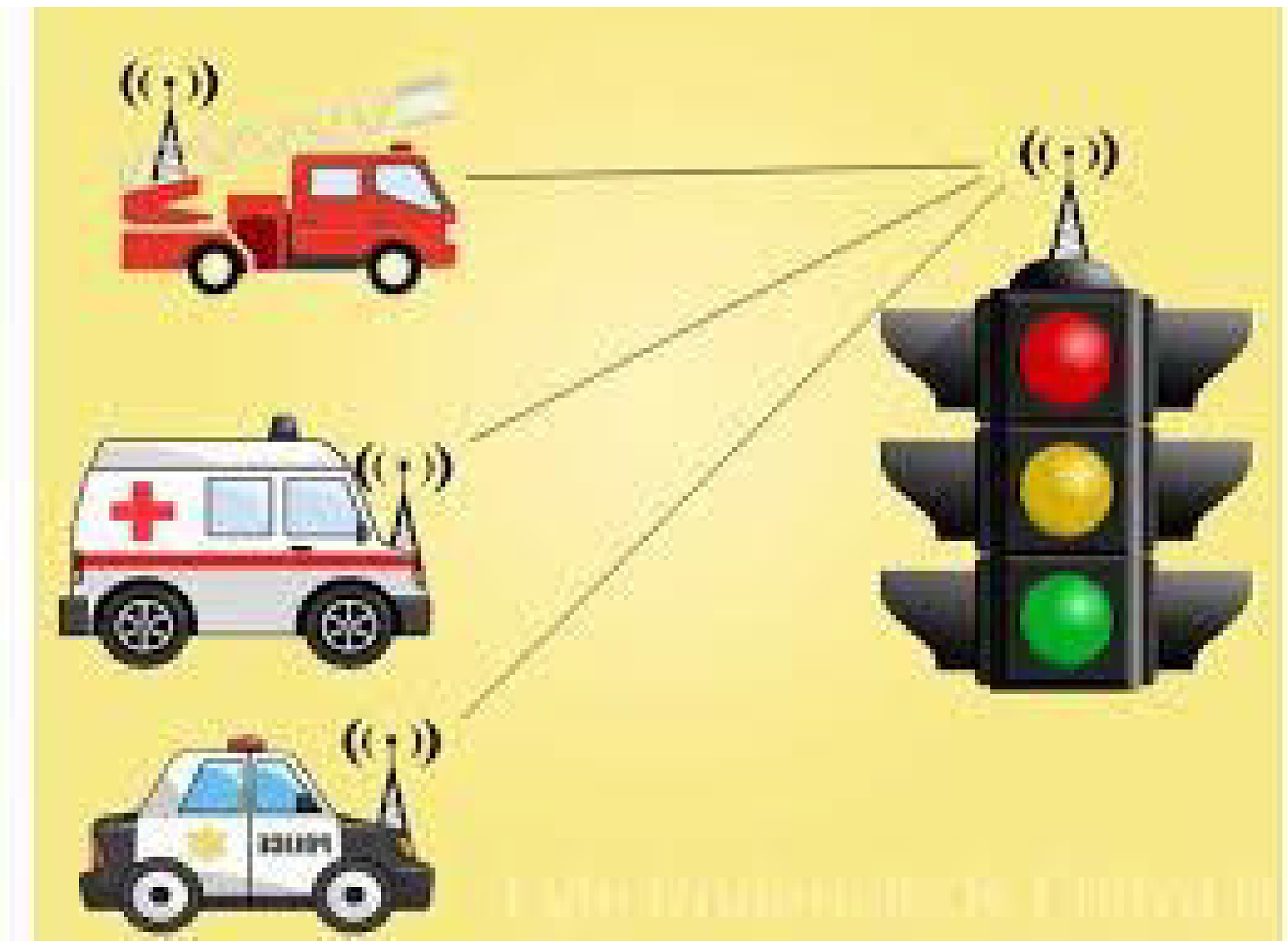


Figure 6.2 v2x wireless communication technology

Used in this way:

- Arduino or Stm32 or AVR to control the traffic light .
- Wireless communication for example (IR Sensor, WIFI) connect between vehicle and traffic light

Steps:

- The traffic signal will operate as usual.
- When the ambulance passes, its location data will be sent to the traffic signal through wireless communication.
- Then, the street where the ambulance is located will be opened.

Advantage:

Using wireless communication technology with a traffic light system to detect an ambulance has several advantages, including:

1. Real-time data: Wireless communication technology can provide real-time data on the location of emergency vehicles, allowing traffic signals to respond quickly and accurately to changing traffic conditions.
2. Improved safety: By detecting emergency vehicles in real time, traffic signals can improve safety by reducing the chance of accidents or collisions between emergency vehicles and other vehicles on the road.
3. Reduced traffic congestion: By responding quickly and accurately to emergency vehicles, traffic signals can help to minimize disruption to traffic flow and reduce congestion on the road.
4. Cost-effective: Using a wireless communication-based system can be more cost-effective than other methods, such as installing sensors in the road or equipping emergency vehicles with expensive communication devices.
5. Compatibility: Wireless communication technology is compatible with a wide range of devices and can be easily integrated with existing traffic signal systems.
6. Scalability: A wireless communication-based system can be easily scaled to cover a larger area or to accommodate more emergency vehicles.

Overall, using wireless communication technology in conjunction with a traffic light system to detect an ambulance can provide more accurate, efficient, and cost-effective traffic management in emergency situations, while also offering greater compatibility and scalability.

Disadvantage:

Using wireless communication technology with a traffic light system to detect an ambulance also has some potential disadvantages, including:

1. Limited range: Wireless communication technology, such as radio frequency or cellular network, may have limited range in some areas, which can affect the accuracy of data transmission.
2. Interference: Wireless signals can be affected by interference from other devices or structures, such as buildings or trees, which can also impact the accuracy of data transmission.
3. Dependence on technology: Wireless communication technology is dependent on the reliability and availability of the network and the devices used, which may not always be guaranteed.
4. Privacy concerns: The use of wireless communication technology may raise privacy concerns among some individuals, as it involves the transmission and storage of data.
5. Maintenance and repair: Wireless communication technology requires regular maintenance and repair to ensure that it is functioning properly, and may be subject to damage or vandalism.

Overall, while using wireless communication technology with a traffic light system to detect an ambulance has several advantages, it also

6.2.3 Using the Sensor and Wireless communication together:

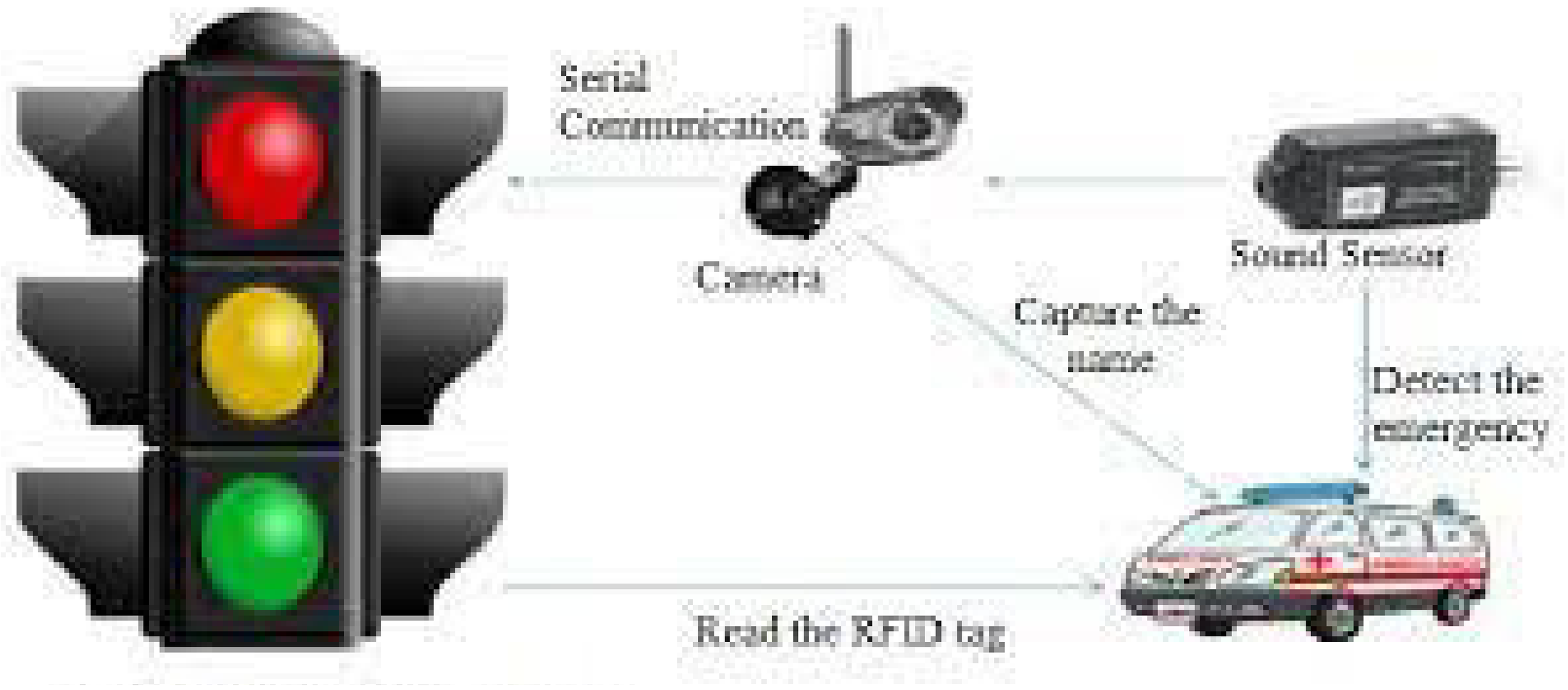


Figure 6.3 traffic signal with RFID reader

In this method, we will use both wireless communication and a camera together to ensure the presence of the ambulance.

Used in this way:

1. Arduino or Stm32 or AVR to control the traffic light .
2. Wireless communication for example (IR Sensor, WIFI) connect between vehicle and traffic light
3. Camera as sensor to detect the ambulance with Raspberry pi

Steps:

1. First, the location of the ambulance will be sent through wireless communication at a certain distance.
2. Secondly, this data will be taken and sent to the Raspberry Pi to verify that it is an ambulance using the camera.
3. Thirdly, when it is verified that it is an ambulance, the data confirming this will be sent.
4. Finally, the traffic signal control unit will unlock the other roads and close the road to the ambulance. (Note: This sentence seems to be the same as the previous one, so I assume it is a mistake and provided the correct translation.

Advantage:

Using a combination of camera and wireless communication technology with a traffic light system to detect an ambulance has several advantages, including:

1. Improved accuracy: A camera can provide more accurate and real-time data on the location of emergency vehicles, while wireless communication technology can transmit this data quickly and accurately to the traffic signal, allowing it to respond in a timely manner.
2. Increased safety: By detecting emergency vehicles in real time, traffic signals can improve safety by reducing the chance of accidents or collisions between emergency vehicles and other vehicles on the road.
3. Better traffic flow: By responding quickly and accurately to emergency vehicles, traffic signals can help to minimize disruption to traffic flow and reduce congestion on the road.
4. Cost-effective: Using a combination of camera and wireless communication-based system can be more cost-effective than other methods, as it can provide accurate and real-time data without requiring expensive sensors or communication devices to be installed on emergency vehicles.
5. Flexibility: A combination of camera and wireless communication-based system is more flexible and can be easily installed in different locations, making it suitable for various traffic conditions and road layouts.

Overall, using a combination of camera and wireless communication technology with a traffic light system to detect an ambulance can provide more accurate, efficient, and cost-effective traffic management in emergency situations, while also offering greater flexibility in implementation

Disadvantage:

Using a combination of camera and wireless communication technology with a traffic light system to detect an ambulance also has some potential disadvantages, including:

1. Dependence on technology: The system is dependent on the reliability and availability of the camera and wireless communication technology, which may not always be guaranteed.
2. Privacy concerns: The use of cameras and wireless communication technology in public areas may raise privacy concerns among some individuals, as it involves the transmission and storage of data.
3. Maintenance and repair: Cameras and wireless communication technology require regular maintenance and repair to ensure that they are functioning properly, and may be subject to damage or vandalism.
4. Complexity: The implementation of a camera and wireless communication-based system may be more complex and require more expertise than other methods, such as installing sensors in the road or equipping emergency vehicles with communication devices.
5. Limited range: Cameras and wireless communication technology may have limited range in some areas, which can affect the accuracy of data transmission.

Overall, while using a combination of camera and wireless communication technology with a traffic light system to detect an ambulance has several advantages, it also has some potential disadvantages that should be considered when implementing such a system.

6.2.4 Using smart internet and cloud technology:

One of the challenges of using smart internet and cloud technology is ensuring that the system is secure and protected from cyberattacks. Another challenge is ensuring that the necessary infrastructure is in place to support the communication, such as high-speed internet and cloud servers.

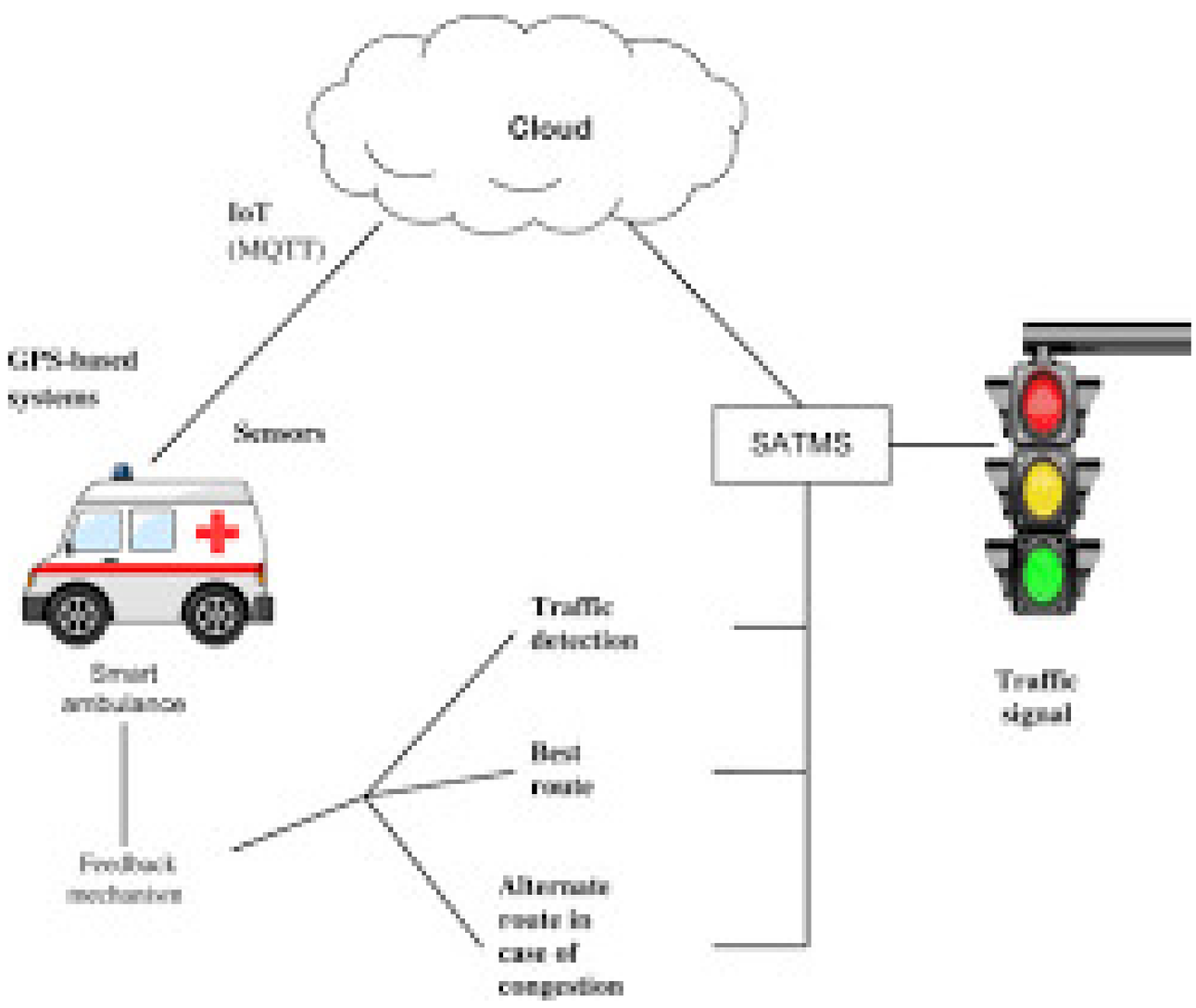


Figure 6.4 smart internet and cloud technology

6.3 Hardware connects:

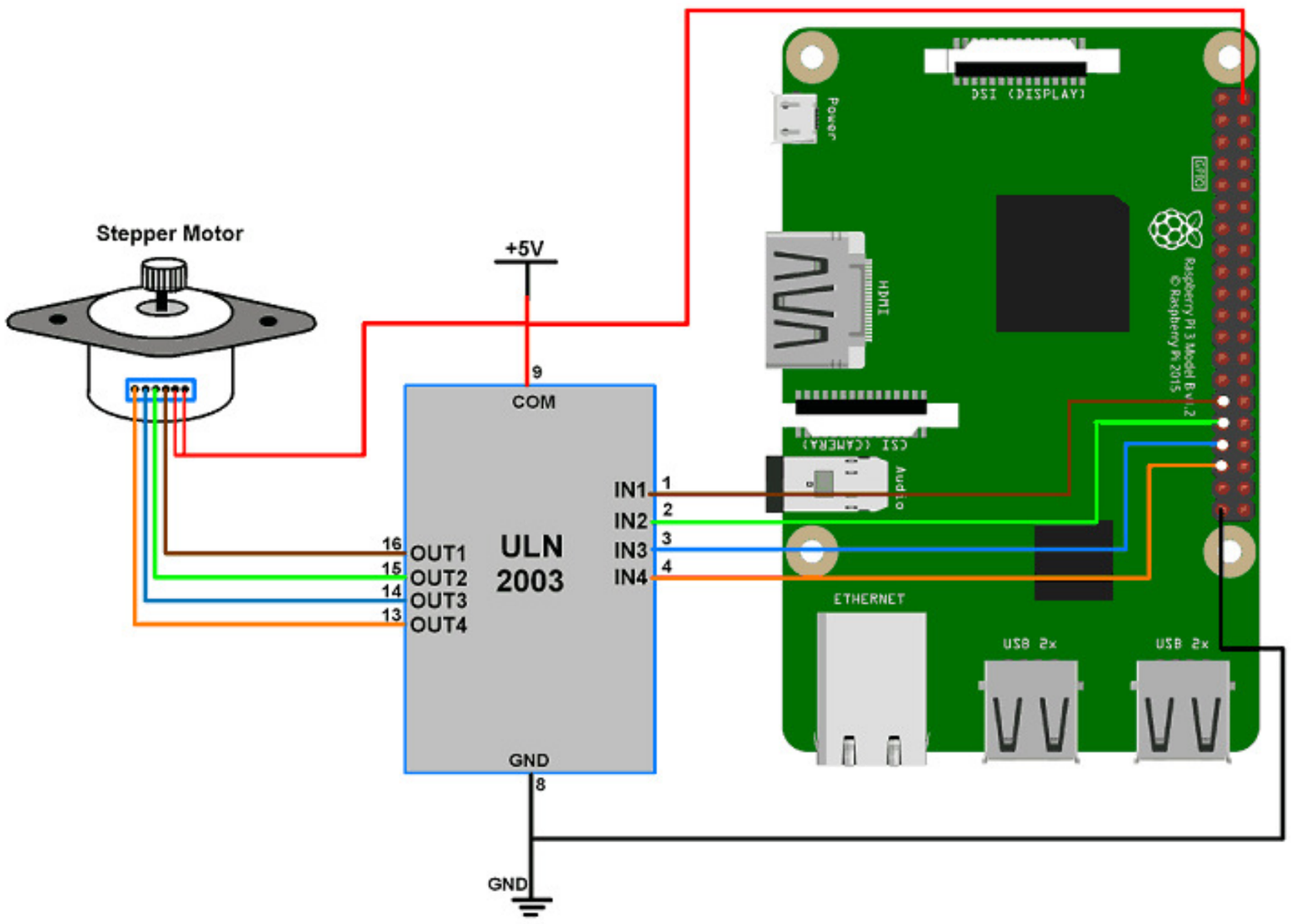


Figure 6.5 Raspberry pi with motor

Figure 6.6 Raspberry pi with Camera

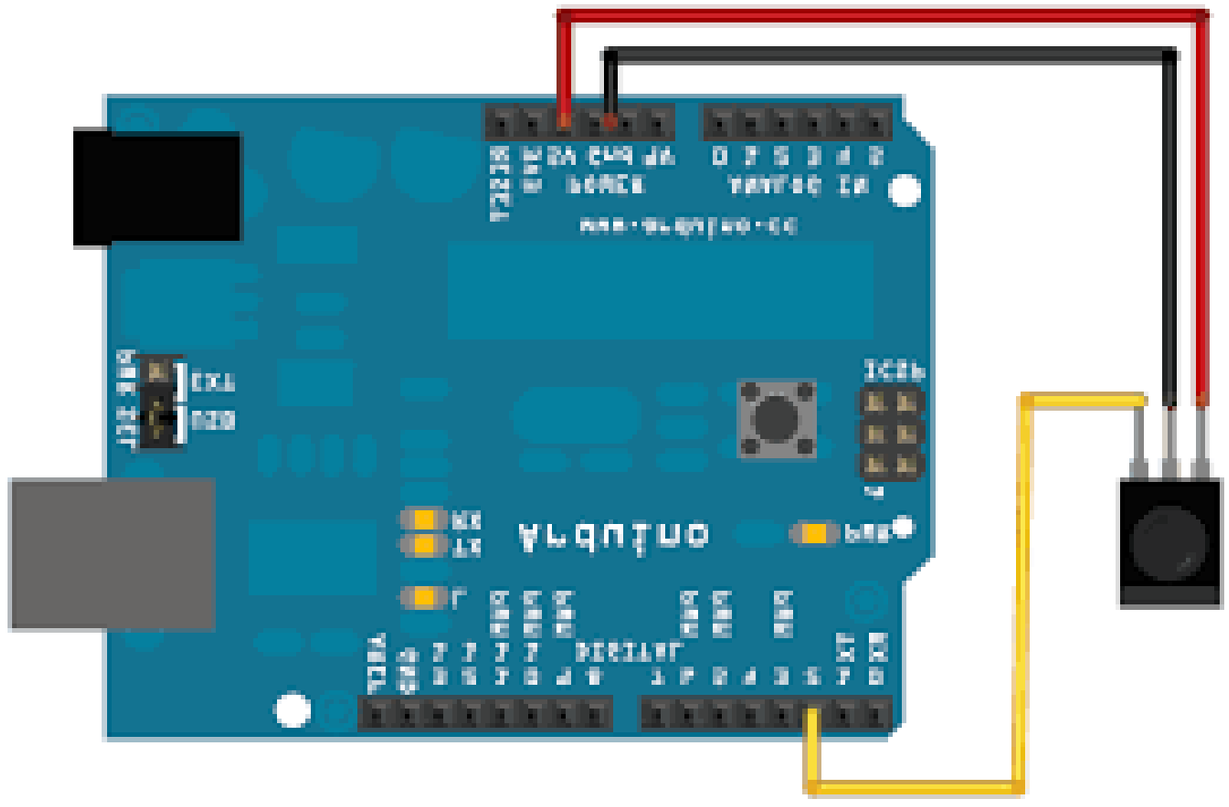


Figure 6.7 Arduino with IR sensor

6.4 Examples of cities that have successfully implemented these systems:

several cities around the world have successfully implemented systems that connect traffic lights to emergency vehicles. Here are some examples:

- Los Angeles, USA: The city of Los Angeles has implemented a system called the Emergency Vehicle Preemption (EVP) system, which allows emergency vehicles to send a signal to traffic lights to request a green light. This system has been in place since the 1970s and has been credited with reducing emergency response times by up to 20%.
- Singapore: Singapore has implemented a system called the Intelligent Transport System (ITS), which includes a network of sensors and communication devices that allow emergency vehicles to communicate with traffic lights. The system can detect the location of emergency vehicles and adjust traffic signals to give them priority.
- Melbourne, Australia: The city of Melbourne has implemented a system called the Priority Traffic Management System (PTMS), which uses GPS technology to track emergency vehicles and adjust traffic signals in real-time. The system can also provide emergency vehicles with optimal routes to their destination.
- London, UK: The city of London has implemented a system called the Traffic Signal Priority system, which uses sensors and communication devices to detect emergency vehicles and adjust traffic signals to give them priority. The system has been credited with reducing emergency response times by up to 40%.
- Amsterdam, Netherlands: The city of Amsterdam has implemented a system called the Green Flow for Emergency Vehicles, which uses a network of sensors and communication devices to detect emergency vehicles and adjust traffic signals to give them priority. The system has been credited with reducing emergency response times by up to 25% [33].

These are just a few examples of cities that have successfully implemented systems that connect traffic lights to emergency vehicles. While each system may have its unique features and challenges, they all share the goal of improving emergency response times and enhancing road safety.

Summary:

- A processing CPU (Central Processing Unit) is the primary component of a computer and the ECM is responsible for monitoring and controlling various systems in the car.
- System inputs are data from raspberry pi MCU, data from Bluetooth module and data from ultrasonic sensors.
- In our project, we deal with cases such as blind spots, forward direction, parking the car and right and left case.
- The blind spot is an area around a vehicle that the driver cannot see directly through the side or rearview mirrors or by turning their head.
- The forward case is the most complicated because it requires a lot of information, including the speed of the forward car and its comparison with its speed, as well as reading the distance sensor.
- The safe distance is between 10 to 15 meters, and it increases by 5 meters for every 10 km/h on highways.
- In the backward case, and only at slow speeds, we can intervene in controlling the car as any damage caused will not result in further damage as in other cases.
- Both autonomous parking and parking assistant features can be helpful tools for drivers when parking their cars. While autonomous parking takes care of the entire parking process, parking assistant provides an additional level of safety by alerting the driver to potential hazards.
- Serial communication is a method of data transfer between two electronic devices by sending and receiving data one bit at a time over a single communication line. In contrast to parallel communication, where multiple data bits are transmitted simultaneously over multiple communication lines, serial communication uses a single line to transmit data.
- Wired communication refers to the transmission of data over a wire-based communication technology. Wired communication is also known as wireline communication.
- Wireless Communication is the fastest growing and most vibrant technological areas in the communication field. Wireless Communication is a method of transmitting information from one point to other, without using any connection like wires, cables or any physical medium.
- Linux is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.
- Linux is like other operating systems such as Windows, macOS, or IOS. Like them, Linux can have a graphical interface and the types of desktop software that you are used to, such as word processors, photo editors, video editors, etc.

- A toolchain is a set of tools that compiles source code into executables that can run on the target device and includes a compiler, a linker, and run-time libraries. Initially, need one to build the other three elements of an embedded Linux system: the bootloader, the kernel, and the root filesystem. It has to be able to compile code written in assembly, C, and C ++ since these are the languages used in the base open source packages
- The kernel is the component that is responsible for managing resources and interfacing with hardware, and so affects almost every aspect of our final software build.
- OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library that provides a wide range of tools and algorithms for analyzing, processing, and manipulating images and videos. It was initially developed by Intel in 1999 and is now maintained by OpenCV.org.
- Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware.
- Image processing is the process of analyzing and manipulating digital images to improve their quality or extract useful information from them. It involves applying mathematical algorithms and techniques to digital images in order to enhance, transform, or extract features from the images.
- The main difference between Haar Cascade and YOLO is that Haar Cascade is a machine learning-based algorithm, while YOLO is a deep learning-based algorithm. Haar Cascade is relatively simple and efficient, but may not be as accurate as YOLO for complex object detection tasks. On the other hand, YOLO is more accurate and can handle complex object detection tasks, but requires more computational resources and is more complex to implement.
- Using wireless communication technology in conjunction with a traffic light system to detect an ambulance can provide more accurate, efficient, and cost-effective traffic management in emergency situations, while also offering greater compatibility and scalability.

Future Work:

- More optimization for the code.
- Fix the lag of displaying data into TFT.
- Improve the sensor algorithms to get more accurate values
- Connect our system to Google Assistant
- Get powerful and suitable DC motor.
- More precise detection
- Better quality camera
- GPS addition for precise location
- Detect heart rate of driver
- Camera detect the accident and send location of nearest hodpital

THANK YOU

Reference:

[1] Ziebinski, Adam, et al. "Review of advanced driver assistance systems (ADAS)." AIP Conference Proceedings. Vol. 1906. No. 1. AIP Publishing, 2017.

[2] Cowan, Nelson. "Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system." Psychological bulletin 104.2 (1988): 163.

[3] Carullo, Alessio, and Marco Parvis. "An ultrasonic sensor for distance measurement in automotive applications." IEEE Sensors journal 1.2 (2001): 143.

[4] Bräunl, Thomas. Embedded robotics. Springer, 2003.

[5] Frenzel, Louis E. Handbook of serial communications interfaces: a comprehensive compendium of serial digital input/output (I/O) standards. Newnes, 2015.

[6] The STM32F401xx devices have an Arm®(a) Cortex®-M4 with FPU and an Arm® Cortex®-A7 with FPU cores.

[7] Hwang, Jong-Young, et al. "Development of novel algorithm and real-time monitoring ambulatory system using Bluetooth module for fall detection in the elderly." The 26th annual international conference of the IEEE engineering in medicine and biology society. Vol. 1. IEEE, 2004.

[8] Golmie, Nada, O. Rebala, and Nicolas Chevrollier. "Bluetooth adaptive frequency hopping and scheduling." IEEE Military Communications Conference, 2003. MILCOM 2003.. Vol. 2. IEEE, 2003.

[9] King, DAVID L., et al. "Applications for infrared imaging equipment in photovoltaic cell, module, and system testing." Conference Record of the Twenty-Eighth IEEE Photovoltaic Specialists Conference-2000 (Cat. No. 00CH37036). IEEE, 2000.

[10] Introduction to Sockets Programming in C using TCP/IP Professor: Panagiota Fatourou TA: Eleftherios Kosmas CSD - May 2012

[11] Anonymous, (n.d), TCP/IP Client and Server [ONLINE] Available: http://pymotw.com/2/socket/tcp.html

[12] "Raspberry Pi Networking Cookbook, second edition" by Rick Golden

[13] Hallinan, Christopher. Embedded Linux primer: a practical real-world approach. Pearson Education India, 2011.

[14] Raghavan, Pichai, Amol Lad, and Sriram Neelakandan. Embedded Linux system design and development. Auerbach Publications, 2005.

[15] Abbott, Doug. Linux for embedded and real-time applications. Elsevier, 2011. 332 | P a g e

[16] Simmonds, Chris. Mastering Embedded Linux Programming. Packt Publishing Ltd, 2017.

[17] Bi, Chun-yue, Yun-peng Liu, and Ren-fang Wang. "Research of key technologies for embedded Linux based on ARM." 2010 International Conference on Computer Application and System Modeling (ICCASM 2010). Vol. 8. IEEE, 2010.

[18] Vaduva, Alexandru, Alex Gonzalez, and Chris Simmonds. Linux: Embedded Development. Packt Publishing Ltd, 2016.

[19] Hallinan, Christopher. "Reducing boot time in embedded linux systems." LINUX journal 2009.188 (2009): 4.

[20] Wang, Ya-Jun. "Research and realization of the mechanism of embedded linux kernel semaphore." 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE). Vol. 1. IEEE, 2010.

[21] Molloy, Derek. Exploring Raspberry Pi: interfacing to the real world with embedded Linux. John Wiley & Sons, 2016.

[22] Simmonds, Chris. Mastering embedded Linux programming. Packt Publishing Ltd, 2015.

[23] Salvador, Otavio, and Daiane Angolini. Embedded Linux Development with Yocto Project. Packt Publishing Ltd, 2014.

[24] Bradski, Gary, and Adrian Kaehler. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[25] Szeliski, Richard. Computer vision: algorithms and applications. Springer Nature, 2022.

[26] Metaxas, Dimitris N. Physics-based deformable models: applications to computer vision, graphics and medical imaging. Vol. 389. Springer Science & Business Media, 2012.

[27] Dollar, Piotr, et al. "Pedestrian detection: An evaluation of the state of the art." IEEE transactions on pattern analysis and machine intelligence 34.4 (2011): 743-761

[28] Gardner, John A., Randy Lundquist, and Steve Sahyun. "TRIANGLE: a tri-modal access program for reading, writing and doing math." Proceedings of the CSUN International Conference on Technology and Persons with Disabilities, Los Angles. Converging Technologies for Improving Human Performance (pre-publication on-line version). Vol. 123. 1998.

.

[30] Armah, Frederick A., David O. Yawson, and Alex ANM Pappoe. "A systems dynamics approach to explore traffic congestion and air pollution link in the city of Accra, Ghana." Sustainability 2.1 (2010): 252-265.

[31] Anand, Jose, and TG Arul Flora. "Emergency traffic management for ambulance using wireless communication." IPASJ International Journal of Electronics & Communication (IIJEC) 2.7 (2014): 43-52.

[32] Sundar, Rajeshwari, Santhoshs Hebbar, and Varaprasad Golla. "Implementing intelligent traffic control system for congestion control, ambulance clearance, and stolen vehicle detection." IEEE sensors journal 15.2 (2014): 1109-1113.

[33] Bibri, Simon Elias, and John Krogstie. "The emerging data–driven Smart City and its innovative applied solutions for sustainability: The cases of London and Barcelona." Energy Informatics 3 (2020): 1-42.

الملخص العربى:

حوادث المرور غير مرغوب فيها، ومع ذلك فهي شائعة إلى حد كبير. وللأسف الشديد، فإننا لم نتجنب الأخطاء المسببة لحوادث الطرق.

يدرك معظم السائقين القواعد العامة وتدابير السلامة عند القيادة، ولكن إهمال بعض السائقين يؤدي إلى وقوع الحوادث. والسبب الرئيسي وراء الحوادث هو الخطأ البشري. وتم توضيح بعض السلوكيات البشرية الشائعة التي تؤدي إلى وقوع الحوادث.

وجد أن ٩٩% من حوادث الطريق يرجع سببها إلى الخطأ البشري، وهو حقيقة مزعجة أن معظم السائقين يواصلون ارتكاب نفس الأخطاء على الطريق دون أن ينتبهوا إلى عادات القيادة السيئة لديهم.

تحدث العديد من الحوادث بسبب الإنشغال عن الطريق، ويمكن لأنظمة المساعدة المتقدمة للسائق والتي تدعي (ADAS) علي تجنب التصادم والحفاظ على المركبة و السيطرة عليها من خلال إصدار إشارات تحذيرية.

تم استخدام الاتصال بين السيارات باستخدام بروتوكول (V2V)، وهو بروتوكول يمكن المركبات من تبادل المعلومات بشكل لاسلكي حول سرعتها وموقعها ومسارها.

كما تم شرح الاتصال بين السيارة وإشارات المرور باستخدام بروتوكول (V2X)، وهو بروتوكول اضيف حديثا، والذي يشمل اتصال المركبات ببعضها البعض بالإضافة إلى اتصال المركبات بالأشياء.

و تم توضيح الاتصال اللاسلكي بين نظام القيادة والمركبة، ومكونات الأجهزة في كل نظام فرعي (نظام قيادة السيارة ونظام معالجة الأوامر والتحكم بها ونظام التحكم بالمواتير)، ووظيفة كل نظام فرعي، والبيانات المتبادلة بينهم.

و تم شرح أيضًا الاتصال اللاسلكي بين سيارتين باستخدام تقنية الواي فاي المدعمة من ال RPI، حيث ان ال RPI هو لوحة تطوير إلكترونية مدعمة من شركة (ARM).

و أيضا تم توضيح المكونات الخارجية المتصلة بال RPI (الكاميرا والسيرفو موتور)، ووظائف ومهام ال RPI، والاتصال التسلسلي بين الأنظمة الفرعية وبعضها، و كذلك وظائف ومهام RPI فيما يتعلق بسيارة الإسعافات الأولية، ووظائف ومهام الأردوينو فيما يتعلق بتقاطعات الطرق، والاتصال التسلسلي بينه وبين ال RPI.

جامعة الأزهر
كلية الهندسة
قسم الألكترونيات و الاتصالات

بكالوريوس. مشروع التخرج
(الإلكترونيات والاتصالات)

نظام متقدم لمساعدة السائق

معتمد على V2V AND V2X

تحت اشراف

د/ محمد يسن ابراهيم عفيفي

قدم بواسطة

محمد رمضان محمد بكر شاهين
بهجت محمود بهجت
عبد الله محمد عبد الله
مريم أحمد سعيد عبد الغفار
اميره محمود بدوي محمود

٢٠٢٣

جامعة الأزهر
كلية الهندسة
قسم الألكترونيات و الاتصالات

بكالوريوس. مشروع التخرج
(الإلكترونيات والاتصالات)

نظام متقدم لمساعدة السائق

معتمد على V2V AND V2X

تحت اشراف

د/ محمد يسن ابراهيم عفيفي

قدم بواسطة

محمد رمضان محمد بكر شاهين
بهجت محمود بهجت
عبد الله محمد عبد الله
مريم أحمد سعيد عبد الغفار
اميره محمود بدوي محمود

٢٠٢٣